General Tree Evaluation for AlphaZero

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Albin Jaldevik

Sequential Decision Making
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

©2024 Albin Jaldevik. All rights reserved.

General Tree Evaluation for AlphaZero

Author: (Rolf) Albin Jaldevik
Student id: 5839408

Abstract

Over the last decade, there have been significant advances in model-based deep reinforcement learning. One of the most successful such algorithms is AlphaZero which combines Monte Carlo Tree Search with deep learning. AlphaZero and its successors commonly describe a unified framework for tree construction and acting. For instance, build the tree with PUCT and act according to visitation counts. Policies based on visitation counts inherently make assumptions about the tree construction. This is problematic since it constrains the construction algorithm. For example, breadth-first tree construction yields a uniform visitation policy. To address this, we investigate the goals when extracting policies from decision trees and propose novel construction decoupled policies. Furthermore, we use these to modify how decision nodes are evaluated and utilize this during tree construction. We support the claim that our novel policies can benefit AlphaZero with theoretical analysis and empirical evidence. Our results on classical Gym environments show that the benefits are especially prominent for limited simulation budgets. The code is available through GitHub[1].

Thesis Committee:

University Supervisor: Wendelin Böhmer, Sequential Decision Making, TU Delft
Committee Member: Neil Yorke-Smith, Algorithmics, TU Delft

[1] https://github.com/albinjal/GeneralAlphaZero

Preface

I would like to express my deepest gratitude towards my supervisor, Wendelin Böhmer, for his guidance and support throughout this thesis. He was the one who introduced me to the subject and suggested the initial research direction which I found very enjoyable. I would also like to thank Neil Yorke-Smith for agreeing to be on my thesis committee and providing feedback. Furthermore, I would like to thank the other members of the Sequential decision-making and Algorithmics research groups for fruitful discussions. In particular PhD student Yaniv Oren and MSc student Felix Kaubek for showing great interest in my work and helping me improve both the report and presentations. Lastly, I would like to thank my friends and family for their unwavering support and encouragement.

Albin Jaldevik
Delft, Netherlands
June 16, 2024

Contents

Preface iii

Contents v

List of Figures vii

1 Introduction 1
- 1.1 Contributions . . . 2
- 1.2 Outline . . . 2

2 Background 3
- 2.1 Reinforcement Learning . . . 3
- 2.2 Monte Carlo Tree Search . . . 4
- 2.3 AlphaZero . . . 6

3 Analysis 9
- 3.1 Tree Evaluation . . . 9
- 3.2 Goals of Tree Evaluation . . . 11
- 3.3 Matrix Notation . . . 12
- 3.4 The Covariance Matrix . . . 15
- 3.5 The Visitation Count Evaluator . . . 18
- 3.6 General Tree Evaluation . . . 23
- 3.7 Modified Construction . . . 29
- 3.8 Numerical Computation . . . 30
- 3.9 AlphaZero . . . 32

4 Related Work 35
- 4.1 Limited Simulation Budget . . . 35
- 4.2 Uncertainty Analysis . . . 36
- 4.3 Non Arithmetic Mean Backup . . . 37

5 Experimental Setup **39**
5.1 Agents 39
5.2 Environments 40
5.3 Value Functions 41

6 Results **45**
6.1 Cliff Walking 45
6.2 Frozen Lakes 52

7 Discussion **55**

8 Conclusion **57**
8.1 Conclusion 57
8.2 Future Work 58

Bibliography **61**

A Implementation Details **67**

B Additional Results **73**

List of Figures

3.1 A decision tree from the perspective of the tree evaluation policy. 11

5.1 6×12 Cliff Walking Environment. The starting state is in the bottom left corner and the goal state is in the bottom right corner. The cliff is located in the bottom row. 41
5.2 Frozen Lake Environments. The starting state is in the top left corner and the goal state is in the bottom right corner. 42
5.3 Heuristic Value Functions used for the Cliff Walking and Frozen Lake. 43

6.1 Cliff Walking. Training and Evaluation mean return over training episodes. The column numbers indicate the simulation budget. 45
6.2 Policy normalized entropy over training episodes. 1 indicates uniform distribution and 0 fully deterministic policy. 46
6.3 Partition into evaluation completion ratio and completion return. The column numbers indicate the simulation budget. 47
6.4 Final evaluation state visitation distribution for the Cliff Walking environment. The row numbers indicate the simulation budget and the columns the agents. . . 48
6.5 Cliff Walking heuristic value function mean return for different simulation budgets (100 seeds). 49
6.6 Cliff Walking heuristic value function average state density (100 seeds). The row numbers indicate the simulation budget and the columns the agents. 49
6.7 Cliff Walking heuristic value function average state density (100 seeds) for baseline (Visit+UCT). The row numbers indicate the simulation budget and the columns the value of the UCT constant c. 50
6.8 Cliff Walking mean tree state counts difference for different UCT constants. Green is for $c = 1$ and red $c = 100$. Each subplot shows a different root state (marked with a black border). 51
6.9 Cliff Walking mean tree state counts difference for different construction algorithms. Blue is MVCUCT and green is regular UCT (baseline). Each subplot shows a different root state (marked with a black border). 52
6.10 4×4 Frozen Lake. Training and Evaluation step performance. 53

6.11 8×8 Frozen Lake. Training and Evaluation step performance. 53
6.12 Final evaluation state visitation distribution for the 4×4 Frozen lake environment. The row numbers indicate the simulation budget and the columns the agents. 54

A.1 Neural Network architecture options. 70
A.2 Neural Network modules. 70

B.1 Cliff Walking 40×12 heuristic value Visit+UCT agent extreme simulation path lengths. 74
B.2 Cliff Walking 40×12 heuristic value Visit+UCT agent extreme simulation state densities for each budgets. The simulation budgets (from left to right) are 16, 64, 256, 1024, and 4096. Stronger red indicates higher density. 74
B.3 4×4 Frozen Lake heuristic value function average state density (100 seeds). The row numbers indicate the simulation budget and the columns the agents. . . 75
B.4 Final evaluation state visitation distribution for the 8×8 Frozen Lake environment. The row numbers indicate the simulation budget and the columns the agents. 76
B.5 8×8 Frozen Lake heuristic value function average state density (100 seeds). The row numbers indicate the simulation budget and the columns the agents. . . 77

Chapter 1

Introduction

Tree search techniques have played a fundamental role in many successful sequential decision-making algorithms over the last century. One early such algorithm is Minimax search [47], which was later extended with alpha-beta pruning [28]. Deep Blue, the first computer program to defeat a world champion in chess in 1997, was based on alpha-beta search [10]. Another important tree search algorithm is Monte Carlo Tree Search (MCTS) which additionally utilizes random rollouts [44].

The last decade has witnessed significant advancements in deep learning, catalyzing breakthroughs across various domains, including computer vision and natural language processing [48]. It also sparked extra incentive for integrating these techniques into sequential decision-making algorithms. Deep reinforcement learning is one such flourishing field [31]. In 2016, Silver et al. [40] published AlphaGo, demonstrating how deep learning can be combined with tree search (MCTS) to achieve superhuman performance in the game of Go. In the following years, the framework was refined into the AlphaGoZero and AlphaZero algorithms [41, 42].

The success of AlphaZero inspired numerous descendants, impacting both practical applications and theoretical advancement. On the practical side, AlphaTensor is an algorithm based on AlphaZero used for discovering novel, faster ways of multiplying matrices [19]. On the theoretical side, MuZero lets the agent learn the environment dynamics as well [39]. Stochastic MuZero supports stochastic environments [1], and Gumbel MuZero improves the performance for large action spaces and low simulation budgets [16, 35].

Despite these advancements, challenges remain. The algorithms in the AlphaZero family commonly describe a unified framework for building and analyzing search trees. For example, AlphaZero uses an algorithm called PUCT for construction and acts under this assumption [38, 42]. If we use another tree construction algorithm than PUCT or even mistune the PUCT hyperparameters, the performance might deteriorate.

This work aims to investigate the possibility of detaching tree construction and evaluation. We hypothesize that novel tree evaluation strategies could, not only allow the application of different construction algorithms but also improve the performance of the default AlphaZero algorithm. Our research aims to address the following questions:

1. How can policies be efficiently extracted from general decision trees?
2. What are the objectives and inherent trade-offs involved in designing tree evaluation policies?
3. Can AlphaZero's performance be improved by diverging from its default tree evaluation policy?
4. Can modifications to tree evaluation and construction boost performance further?

The first two questions are answered analytically in Chapter 3, while the other two are answered empirically in Chapter 6. Next, we provide a summary of our contributions and an outline of the thesis.

1.1 Contributions

This thesis's contributions are summarized as follows:

- We propose a framework for general MCTS tree evaluation based on value estimation.
- We parametrize the value estimator and provide insight into its variance.
- We propose multiple concrete tree evaluators balancing expectation and variance.
- We show how our framework could also be used to improve tree construction.
- We empirically demonstrate that our framework, using one of the proposed evaluators, outperforms the AlphaZero baseline in certain environments. This holds both when only modifying the evaluation and when modifying both evaluation and construction.
- We investigate and provide insight into why performance is improved.
- We suggest several directions for future work in this domain.
- We provide an open-source modular implementation of our AlphaZero framework.

1.2 Outline

The remainder of this thesis is structured as follows. In Chapter 2, we provide the necessary background on reinforcement learning, MCTS, and AlphaZero. Chapter 3 is the main analytical part of the thesis describing our proposed framework for general tree evaluation. We provide theoretical results and propose several novel policies. The chapter also discussed practical computation aspects as well as modifications to the construction algorithms. Chapter 5 describes the experiments conducted. The experiments evaluate the performance of the novel policies compared to the baseline. Chapter 6 reports and comments on the results of the experiments. In Chapter 7 the experimental results are discussed and analyzed. Lastly, Chapter 8 describes our conclusion and discusses directions for future work.

Chapter 2

Background

2.1 Reinforcement Learning

In reinforcement learning (RL), the goal is to learn a *good* policy (agent) $\pi(a|s)$ by interacting with an environment. The policy $\pi(a|s)$ denote the probability of taking action a in state s. The environment is traditionally represented as a Markov Decision Process (MDP) [4] defined by a tuple $\langle \mathcal{S}, \mathcal{A}, \rho, \mathcal{P}, \mathcal{R} \rangle$. $\mathcal{S}$ represent the set of states, $\mathcal{A}$ the set of actions, ρ the initial state distribution, $\mathcal{P}$ the transition model (probability of transitioning to a new state given the current state and action), and $\mathcal{R}$ the reward distribution. We will primarily focus on deterministic environments which simplifies ρ, $\mathcal{P}$, and $\mathcal{R}$. We can also include a discount factor $\gamma \in (0,1)$ indicating how we value future rewards. We define the value $V_\pi(s)$ of a policy π as the discounted expected cumulative reward (return) starting in state s

$$V_\pi(s) = \mathbb{E}[\sum_{t=0}^{H} \gamma^t r_t \mid r_t \sim \mathcal{R}(s_t, a_t, s_{t+1}), s_{t+1} \sim \mathcal{P}(s_t, a_t), a_t \sim \pi(s_t), s_0 = s].$$

Note that the rewards, states, and actions are sequentially sampled. We will refer to this sampled sequence as a trajectory. The sampling stops when we reach a terminal state in the environment or after an optional maximal time horizon H (we can also let $H = \infty$). The goal in reinforcement learning is usually to find an optimal policy π_* that maximizes the value function $V_{\pi_*}(s_0)$ when $s_0 \sim \rho$ [43]:

$$\pi_* = \arg\max_\pi \mathbb{E}[V_\pi(s_0) \mid s_0 \sim \rho].$$

Note that there could exist multiple optimal policies. It is often useful to expand the value function into a recursive form

$$V_\pi(s) = \mathbb{E}[r + \gamma V_\pi(s') \mid r \sim \mathcal{R}(s,a,s'), s' \sim \mathcal{P}(s,a), a \sim \pi(s)] = \mathbb{E}[Q_\pi(s,a) \mid a \sim \pi(s)] \quad (2.1)$$

where $Q^\pi(s,a)$ is the expected discounted return starting in state s, taking action a, and then following policy π. The recursive formulation in equation 2.1 is commonly referred to as

one of the Bellman equations [4]. From this recursive formulation, we can conclude that an optimal policy π_* must satisfy the Bellman optimality equation

$$V_{\pi_*}(s) = \mathbb{E}[r + \gamma V_{\pi_*}(s') \mid r \sim \mathcal{R}(s,a,s'), s' \sim \mathcal{P}(s,a), a \sim \pi_*(r)],\ \forall s \in \mathcal{S}.$$

that is, the optimal policy is the policy that maximizes the expected reward plus the expected value of the next state following the optimal policy. This lets us consider the optimal policy on a step-by-step basis. This comes down to optimization over the possible actions in the current state.

In a reinforcement learning context, the full MDP is not always provided but instead, the goal is to learn a good policy solely from interacting with the environment (sequential sampling from the MDP).

2.2 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a popular heuristic search algorithm for sequential decision-making problems. It combines ideas from decision tree search used by methods like Minimax search with elements of Monte Carlo sampling [44]. A decision tree is a tree where each node represents a unique action sequence, and the root node usually represents the current state (no action taken). In some literature, the nodes represent states but this is less flexible since it can potentially cause issues when it comes to stochastic or cyclic environments.

The main idea in MCTS is to build the tree by sequentially sampling from the environment and then using these samples to guide the search. The algorithm is commonly divided into four main steps. First, *select* a node in the tree to expand. Second, *expand* the selected node by adding a new node (execute a new action in the environment). Third, *simulate* a rollout from the new node to the end of the episode to estimate the value of the node. Finally, *backup* the value estimate to the root node. The algorithm repeats these steps until a stopping criterion such as time or compute is met. In this work, the stopping criterion is *simulation budget* which is the number of repetitions of the four steps. The budget can also be thought of as the number of visits to the root node or the number of nodes in the tree. When the search stops, we evaluate the tree to determine the policy at the root. Algorithm 1 describes the high-level structure of how a policy is extracted with the MCTS algorithm.

Algorithm 1 MCTS: Monte Carlo Tree Search

Require: MDP, SimulationBudget, s
1: Tree ← **ConstructTree**(MDP, SimulationBudget, s)
2: **return ExtractPolicy**(Tree)

The construction of the decision tree is a vital component of the algorithm. Algorithm 2 describes how the MCTS tree is built. The **Select** function in Algorithm 2 is responsible for selecting a node in the tree to expand. This is commonly done by iterative sampling from a selection policy such as UCT (Upper Confidence Trees) [29] until an expandable node is

Algorithm 2 ConstructTree

Require: MDP, SimulationBudget, s
1: Tree $\leftarrow$ **Node**(s)
2: $i \leftarrow 0$
3: **while** $i <$ SimulationBudget **do**
4: Node $\leftarrow$ **Select**(Tree)
5: NewNode $\leftarrow$ **Expand**(MDP, Node)
6: $v \leftarrow$ **Evaluate**(NewNode)
7: NewNode.$v \leftarrow v$
8: **Backup**(NewNode)
9: $i \leftarrow i+1$
10: **end while**
11: **return** Tree

reached. UCT is based on the Upper Confidence Bound algorithm (UCB1) [2]. The UCT policy is defined as

$$\pi_{\text{UCT}} = \underset{\mathcal{A}}{\text{PolicyMax}}\,[\text{UCT}]$$

with

$$\underset{\mathcal{A}}{\text{PolicyMax}}\,[f]\,(x,a) = \frac{\delta\left(f(x,a) - \max_{a' \in \mathcal{A}} f(x,a')\right)}{\int_{\mathcal{A}} \delta\left(f(x,a'') - \max_{a' \in \mathcal{A}} f(x,a')\right) da''} \propto \delta\left(f(x,a) - \max_{a' \in \mathcal{A}} f(x,a')\right) \tag{2.2}$$

where the integral is a sum in the discrete case and δ is the Dirac delta function defined as

$$\delta(x) = \begin{cases} 1 & \text{if } x = 0, \\ 0 & \text{otherwise.} \end{cases}$$

The UCT score is defined as

$$\text{UCT}(x,a) = \bar{Q}(x \uplus a) + c \cdot \sqrt{\frac{\log N(x)}{N(x \uplus a)}}.$$

The PolicyMax function returns a uniform distribution over the actions with the highest value. The UCT function gives the UCT score of each child node. The notation $x \uplus a$ denotes the node we arrive at by taking action a from node x. The function $\bar{Q}(x)$ returns the empirical mean return of collected trajectories that start in node x. The constant c is a positive hyperparameter and $N(x)$ denotes the visitation count for node x. If we reach a node that is not fully expanded we select it for expansion.

After selecting a node, we **Expand** it by taking a new action (sampled uniformly among the unexpanded) in the environment, this adds a new node to the tree. We then **Evaluate** to get a bootstrap estimate for its value. In the default version of MCTS, simulation is done by sequentially sampling a finite horizon trajectory from the MDP with some rollout policy and returning its discounted return. The rollout policy is usually the uniform distribution.

Finally, we **Backup** the value estimate to the root node by updating $\bar{Q}$ and N for all nodes in the path from the root to the new node. The backup step is described in Algorithm 3. Node.r is the reward observed in the environment when expanding the node. Node.N is initialized to 0 but incremented during backup. For the new node, line 6 will simply assign Return to Node.Q since Node.N is 1. The final step in the MCTS algorithm is to **ExtractPolicy**

Algorithm 3 Backup

Require: NewNode, γ
1: Node ← NewNode
2: Return ← Node.v
3: **while** Node is not None **do**
4: Node.N ← Node.N + 1
5: Return ← Node.r + $\gamma *$ Return
6: Node.Q ← Node.Q + (Return - Node.Q) / Node.N
7: Node ← Node.Parent
8: **end while**

from the constructed tree. This is done by selecting the action at the root with the highest visitation count. The deterministic visitation count policy is defined as

$$\pi_{N'} = \underset{\mathcal{A}}{\text{PolicyMax}} \left[N(x \uplus a) \right].$$

Simply selecting the action with the highest visitation count makes the most sense if we use some kind of optimistic tree construction algorithm such as UCT. The exploration term in UCT will decay for large numbers of visits which means that the policy will converge to selecting the action with the highest value $\bar{Q}$. This is proved in Section 3.5.

2.3 AlphaZero

AlphaZero (AZ) is a framework for combining MCTS with elements of deep learning. Its main predecessor AlphaGo was published by Silver et al. [40] (DeepMind) in 2016. The framework was extended in a series of papers to improve performance and generality, and in late 2017, the version called AlphaZero was published [41, 42].

The core component of the algorithm involves iteratively collecting experiences through self-play and learning from them. An overview of the algorithm is described in Algorithm 4. Note that there is a fundamental difference between MCTS and AlphaZero in that MCTS returns a policy for a specific state while AlphaZero returns a trained neural network (NN). This neural network is then used to determine future policies. MCTS is a search algorithm while AZ is also a framework for learning with modified MCTS as its core component. The rest of this section will be for investigating the algorithm components in further detail.

Algorithm 4 AZ: AlphaZero

Require: MDP, SimulationBudget, BatchSize, Iterations, NN
1: ReplayBuffer $\leftarrow$ **ReplayBuffer**()
2: $i \leftarrow 0$
3: **while** $i <$ Iterations **do**
4: Trajectories $\leftarrow$ **CollectTrajectories**(MDP, SimulationBudget, BatchSize, NN)
5: ReplayBuffer.**extend**(Trajectories)
6: **Learn**(NN, ReplayBuffer)
7: $i \leftarrow i+1$
8: **end while**
9: **return** NN

2.3.1 AlphaZero MCTS

The trajectories collected by **CollectTrajectories** are sequentially sampled from the MDP using the AlphaZero Monte Carlo Tree Search (AZMCTS) policy at each environment step. In a multi-player setting, such as in the game of Go, the trajectories are collected through self-play. The AZMCTS policy is generated through a modified version of MCTS (Algorithm 1) which uses the neural network during **Evaluate** instead of random rollouts.

The neural network in AlphaZero, parametrized by θ, maps an environment state s to a value v_θ and a policy π_θ. The policy π_θ is sometimes referred to as the *prior policy*. Further details on state embeddings and architectures used in this work are described in Appendix A.

AlphaZero presents three main modifications to the default MCTS algorithm. The first modification is that during tree construction, v returned from **Evaluate** is replaced by v_θ from the neural network. This value will still be referred to as a simulation value. **Evaluate** additionally returns the prior policy π_θ which is stored for each node in the tree.

The second modification that AlphaZero makes, is that it uses PUCT (Predictor + UCT) as the selection policy rather than UCT [38, 42]. PUCT uses the prior policy produced by the neural network for more sophisticated exploration. The PUCT selection policy is defined as

$$\pi_{\text{PUCT}} = \underset{\mathcal{A}}{\text{PolicyMax}}\,[\text{PUCT}]$$

with

$$\text{PUCT}(x,a) = \bar{Q}(x \uplus a) + c \cdot \pi_\theta(x,a) \cdot \frac{\sqrt{N(x)}}{1+N(x \uplus a)}$$

where c is a positive constant. PUCT is based on many of the same principles as UCT. It prefers actions with larger $\bar{Q}$ and gives a bonus to actions with low visitation counts.

The last modification in AZMCTS is that during learning, **ExtractPolicy** returns a stochastic policy proportional to the visitations instead of just taking the max. This is primarily done for exploration purposes since it increases the diversity in trajectories to learn from.

The visitation count policy used in AZMCTS is defined as

$$\pi_N(x,a) = \frac{N(x \uplus a)}{N(x)} \propto N(x \uplus a).$$

Note that $N(x \uplus a) = 0$ for unexpanded actions. If $N(x) = 0$ we return a uniform distribution. During evaluation, the deterministic visitation count policy is used.

2.3.2 Learning

During the learning phase, AlphaZero aims to improve the parameters θ of the neural network based on previous experiences. The parameters are updated iteratively using stochastic gradient descent to minimize the loss function. The loss function in AlphaZero is of the form

$$L_{\text{AZ}} = \alpha L_{\text{Value}} + (1-\alpha) L_{\text{Policy}}$$

for $\alpha \in (0,1)$. We can also add an optional regularization term to the loss function. The value and policy losses are computed over a batch of trajectories T sampled from the replay buffer. The value loss is computed as the mean squared error between the current value network output and the target value

$$L_{\text{Value}} = \| v_\theta(T[s]) - v_{\text{target}}(T) \|^2$$

where $v_\theta(T[s])$ is a vector of the neural network outputs for the states in the trajectory and $v_{\text{target}}(T)$ is a vector with the value targets for those states. The norm is the Euclidian l^2-norm. In the default AlphaZero implementation, we use n-step bootstrapped value targets [31]:

$$\left[v_{\text{target}}(T)\right]_t = \sum_{i=0}^{n-1} \gamma^i \cdot T[r]_{t+i} + \gamma^n \cdot v_\theta(T[s])_{t+n} \cdot \neg T[\text{terminated}]_{t+n}$$

where $T[r]$ is a vector of rewards, γ the discount factor, and n the number of steps to bootstrap. The entries of $T[\text{terminated}]$ is 1 if the trajectory has terminated. Note that we do not let any gradient flow from the value target into the value loss.

The policy loss is decreased if the policy network successfully predicts the prior policies π_{target}, extracted by AZMCTS. The loss for a batch of sampled trajectories T is a mean over each timestep where the policy is evaluated

$$L_{\text{Policy}} = \frac{1}{|T|} \sum_{(\pi_{\text{target}}, s) \in T} \sum_{a \in \mathcal{A}} -\pi_{\text{target}}(s,a) \cdot \log \pi_\theta(s,a)$$

where $|T|$ is the cardinality including both the learning batch size as well as the trajectory lengths.

This concludes our description of the AlphaZero algorithm. More details on the implementation and hyperparameters used in this work can be found in Appendix A. Now with the necessary background, we can move on to the main part of the thesis where we propose a framework for general tree evaluation.

Chapter 3

Analysis

Let's consider the task of determining a *good* policy π from an arbitrary decision tree. Each node in the tree represents a unique sequence of actions and records the reward obtained when taking that action. In a stochastic environment, the reward can be modeled as a sample from a random variable conditioned on the action sequence. In the MCTS framework, each node contains a simulation value which we can model as a sample from a random variable conditioned on the state. In traditional MCTS this sample comes from a random rollout (with aleatoric uncertainty) but in AlphaZero it comes from a neural network, where the uncertainty is epistemic.

3.1 Tree Evaluation

As mentioned in the previous chapter, the goal is to find a policy π_* that maximizes the value of the current state $V_{\pi_*}(s)$. The classical Bellman equations recursively define the value of a certain state. When working with decision trees, we want to consider the value of a node in the tree instead. We use x to denote a node in the tree and $\mathcal{T}_x$ to denote all nodes in the subtree rooted at x. If the transition model of the MDP is stochastic, the state and reward observed in each tree node are random variables. Each node in the tree is associated with a unique action sequence from the root state. If we know the transition model $\mathcal{P}$, we can determine the probability of observing a state in each node. Rigorously, we can recursively define the state distribution of each node in the tree. Let $f_{\mathcal{P}(s,a)}(s')$ be the probability density (mass in the discrete case) function of the transition model $\mathcal{P}(s,a)$. Then the state distribution of a node $x \uplus a$ is given by

$$f_{S(x \uplus a)}(s') = \int_{\mathcal{S}} f_{\mathcal{P}(s,a)}(s') f_{S(x)}(s) ds.$$

The integral is a sum over $\mathcal{S}$ in the discrete case. A standard assumption is that we know the state of the root node $S(x) = s_0$. This implies that we can recursively determine the state distribution of all nodes in the tree. We can also consider the probability of a node being terminal in a similar fashion.

We can additionally consider the distribution of rewards $R(x)$ when entering a node x. The distribution is dependent on the state distribution of the parent, child, and the reward dy-

namics of the environment. Marginalizing over the state distribution of the parent and child gives the reward distribution of the node

$$f_{R(x \uplus a)}(r) = \int_S \int_S f_{\mathcal{R}(s,a,s')} f_{S(x \uplus a)}(s') f_{S(x)}(s) ds ds'.$$

In the discrete case, the integrals are replaced with summations. Note that if the environment transition model is deterministic, the difference between R and $\mathcal{R}$ diminishes.

The first step in our analysis is to convert the Bellman equations (2.1) to the decision tree framework by defining the value of a decision node as

$$V_\pi(x) = \mathbb{E}[Q_\pi(x \uplus a) \mid a \sim \pi(x)]$$

where

$$Q_\pi(x) = \mathbb{E}[R(x)] + \gamma V_\pi(x).$$

Traditionally the Q value is parameterized by state and action but we now consider the Q of a node in the tree instead ($Q(x,a) = Q(x \uplus a)$). Similarly, $\pi(x)$ is a policy determined from a tree node x instead of a state. There could be multiple nodes with the same state yielding different policies. In stochastic environments, we do not know the exact expected reward $\mathbb{E}[R(x)]$ but we can estimate it with the obtained sample $r(x)$. In deterministic environments, the observed reward $r(x)$ is precisely the expected reward $\mathbb{E}[R(x)]$. This thesis primarily focuses on deterministic environments since this is the classical setting for AlphaZero (Chess, Go, Atari, etc). This simplifies some parts of the analysis but in theory, the framework should work for stochastic environments as well.

To find the optimal value of a node we would have to expand these recursive formulations until all leaves reach terminal states since in the deterministic setting, the value of terminal nodes is zero. This is possible for some simpler environments like Tic-Tac Toe but it is not feasible for more complex environments like Chess or Go. Instead, MCTS builds on incorporating the simulation values $v(x)$ to estimate the value of a node

$$\hat{V}_{\tilde{\pi}}(x) = \tilde{\pi}(x,a_v)v(x) + \sum_{a \in \mathcal{A}} \tilde{\pi}(x,a)\hat{Q}_{\tilde{\pi}}(x \uplus a) = \sum_{a \in \mathcal{A}_v} \tilde{\pi}(x,a)\hat{Q}_{\tilde{\pi}}(x \uplus a) \tag{3.1}$$

and

$$\hat{Q}_{\tilde{\pi}}(x) = r(x) + \gamma \hat{V}_{\tilde{\pi}}(x) \tag{3.2}$$

where $\mathcal{A}_v = \mathcal{A} \cup a_v$ and $\hat{Q}(x \uplus a_v) = v(x)$. This notation is convenient since we can think of the simulation as a special action and $\tilde{\pi}$ as a tree evaluation policy incorporating it. This gives us a new framework for analyzing tree policies. Note that $\tilde{\pi}$ has the properties of a proper policy but we can not take the action a_v in the real environment. When acting in the real environment we can convert the tree evaluation policy into a real policy by

$$\pi(x,a) = \frac{\tilde{\pi}(x,a)}{1 - \tilde{\pi}(x,a_v)} \propto \tilde{\pi}(x,a).$$

Figure 3.1 shows how we can think of v as special action nodes in the decision tree. We can consider v as a noisy measurement of some value function $V_\pi(x)$. For example, in default MCTS, π is the rollout policy.

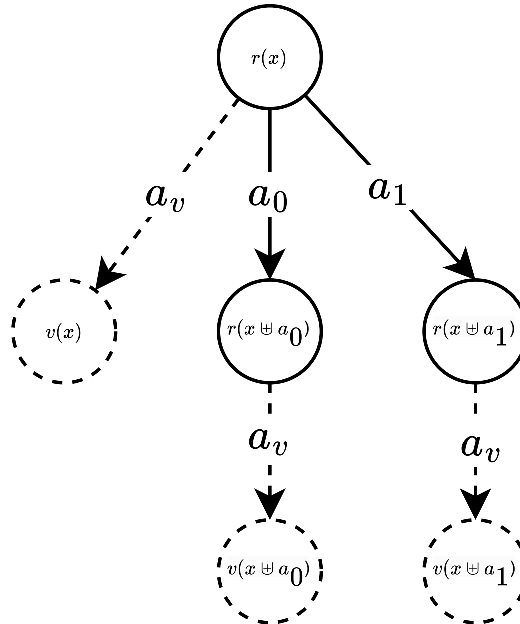


Figure 3.1: A decision tree from the perspective of the tree evaluation policy.

3.2 Goals of Tree Evaluation

In the previous section, we established that we can use a tree evaluation policy to estimate the value of each node in a tree. In this section, we investigate the desirable properties of such value estimates. We can think of the tree evaluation policy as a parameterization of the value estimator. The goal of this estimator is to provide the most accurate estimate of the optimal value of the root nodes' children since this would allow us to extract the optimal policy. We also generally want to increase the expected value estimate at the root since it should correspond to a higher-value policy.

A good statistical estimator should have low bias and low variance. Bias in the value estimator arises if the expected value of the estimator deviates from the optimal value. In this context, the estimator might underestimate the optimal value since the value of all nonoptimal policies is lower. To counteract this, we could find the tree evaluation policy that maximizes the value

$$\tilde{\pi}_Q(x) = \arg\max_{\tilde{\pi}} \hat{V}_{\tilde{\pi}}(x).$$

We will refer to this evaluator as the Q-evaluator[1] which simply picks the action with the highest Q-value estimate

$$\tilde{\pi}_Q(x,a) = \operatorname*{PolicyMax}_{\mathcal{A}_v} \left[\hat{Q}_{\tilde{\pi}}(x \uplus a)\right].$$

[1] The implementation might differ depending on whether nodes are fully expanded or not. See Appendix A.

This is a similar policy to the one used in Q-learning as well as in Minimax tree search [51, 47]. The problem is that $\hat{Q}_{\tilde{\pi}}$ is also a random variable and could require extensive search to be sufficiently sure about. These are empirically estimated Q-values that introduce variance.

The variance in the value estimate arises from the uncertainty in the simulation values as well as the rewards in stochastic environments. The variance of the estimator is given by

$$\mathbb{V}[\hat{V}_{\tilde{\pi}}(x)] = \sum_{(a,b)\in\mathcal{A}_v\times\mathcal{A}_v} \tilde{\pi}(x,a)\tilde{\pi}(x,b)\,\mathrm{Cov}[\hat{Q}_{\tilde{\pi}}(x \uplus a), \hat{Q}_{\tilde{\pi}}(x \uplus b)].$$

Intuitively, the variance will therefore be decreased if the tree evaluator prioritizes actions with low variance as well as spreading the policy over actions with low covariance. The Q-evaluator does not take the variance into account and therefore might have a high variance.

3.3 Matrix Notation

Before proceeding to the main results, we devote this section to some convenient matrix notation. We can define vectors by mapping sets instead of individual elements. For example, let

$$x \uplus \mathcal{A} = \{x \uplus a \mid a \in \mathcal{A}\}$$

and

$$\tilde{\pi}(x,\mathcal{A}_v) = \left[\tilde{\pi}(x,a) \mid a \in \mathcal{A}_v\right].$$

In this notation, the value estimate can be written as

$$\hat{V}_{\tilde{\pi}}(x) = \tilde{\pi}(x,\mathcal{A}_v)^T \hat{Q}_{\tilde{\pi}}(x \uplus \mathcal{A}_v)$$

with its variance given by

$$\mathbb{V}[\hat{V}_{\tilde{\pi}}(x)] = \tilde{\pi}(x,\mathcal{A}_v)^T \mathrm{Cov}[\hat{Q}_{\tilde{\pi}}(x \uplus \mathcal{A}_v)]\tilde{\pi}(x,\mathcal{A}_v).$$

The beauty of this notation is that we can also use the set of nodes in the subtree $\mathcal{T}_x$ to expand the estimators for the full subtree under the assumption that $\tilde{\pi}(x,a) = 0$ for non-expanded child nodes. Define the discounted probability of reaching node y from node x as

$$\Lambda_{\tilde{\pi}}(x,y,\gamma) = \prod_{(z,a)\in\mathrm{path}(x,y)} \gamma\tilde{\pi}(z,a)$$

where

$$\mathrm{path}(x,y) = \{\,(x,a_0),(x \uplus a_0,a_1),\ldots,(x \uplus a_0 \uplus \ldots \uplus a_{n-1},a_n)\,\}$$

so that $x \uplus a_0 \uplus \ldots \uplus a_n = y$. Note that $\mathrm{path}(x,x) = \emptyset$ so $\Lambda_{\tilde{\pi}}(x,x,\gamma) = 1$. We can now expand the Q-value estimate to

$$\hat{Q}_{\tilde{\pi}}(x) = \Lambda_{\tilde{\pi}}(x,\mathcal{T}_x,\gamma)^T \left(r(\mathcal{T}_x) + \gamma\tilde{\pi}(\mathcal{T}_x,a_v) \odot v(\mathcal{T}_x)\right) \tag{3.3}$$

where $\odot$ is the element-wise product and $x \in \mathcal{T}_x$ (the root is in the subtree). This can also be expressed as

$$\hat{Q}_{\tilde{\pi}}(x) = \Lambda_{\tilde{\pi}}(x, \mathcal{T}_x, \gamma)^T r(\mathcal{T}_x) + \Lambda_{\tilde{\pi}}(x, \mathcal{T}_x \uplus a_v, \gamma)^T v(\mathcal{T}_x).$$

Proof (3.3). One can utilize proof by induction to show that the matrix (3.3) and recursive formulations are equivalent (3.2). Let $l(x) \in \{0, 1, \ldots, n\}$ denote the depth level in the tree of the node x, that is, its distance from the root. The root node is the only node at level 0. The proof includes two steps: the base case and the induction step. In the base case, we show that the formulation holds for all nodes at level n. In the induction step, we assume that the formulation holds for all nodes at level n and show that it also holds for all nodes at level $n-1$.

Base case: $l(x) = n$. Since n is the last level in the tree, all nodes at this level must be leaf nodes. By definition, a leaf has no children and we assume that $\tilde{\pi}(x,a) = 0 \ \forall a \in \mathcal{A}$. For a leaf x, the tree is simply the node itself

$$\mathcal{T}_x = \{x\} \text{ if } l(x) = n.$$

which implies that the Q-value estimate is

$$\begin{aligned} \hat{Q}_{\tilde{\pi}}(x) = \\ \Lambda_{\tilde{\pi}}(x, \mathcal{T}_x, \gamma)^T \left(r(\mathcal{T}_x) + \gamma\tilde{\pi}(\mathcal{T}_x, a_v) \odot v(\mathcal{T}_x)\right) = \\ \Lambda_{\tilde{\pi}}(x, \{x\}, \gamma)^T \left(r(\{x\}) + \gamma\tilde{\pi}(\{x\}, a_v) \odot v(\{x\})\right) = \\ \Lambda_{\tilde{\pi}}(x, x, \gamma) \left(r(x) + \gamma\tilde{\pi}(x, a_v) v(x)\right) = \\ r(x) + \gamma v(x) = \\ r(x) + \gamma \hat{V}_{\tilde{\pi}}(x) \end{aligned}$$

since $\Lambda_{\tilde{\pi}}(x,x,\gamma) = 1$ and $\tilde{\pi}(x, a_v) = 1$ for leaf nodes. This shows that (3.3) is equivalent to (3.2) for all nodes in level n.

Induction step: $l(x) = L-1$. Suppose that (3.3) holds for all nodes at level L. We will show that it also holds for all nodes at level $L-1$. By definition (3.2), the value of a node at level $L-1$ is given by

$$\hat{Q}_{\tilde{\pi}}(x) = r(x) + \gamma\tilde{\pi}(x, a_v) v(x) + \gamma \sum_{a \in \mathcal{A}} \tilde{\pi}(x,a) \hat{Q}_{\tilde{\pi}}(x \uplus a)$$

where $Q_{\tilde{\pi}}(x \uplus a)$ are at level L. We can expand this expression using the induction

hypothesis for $\hat{Q}_{\tilde{\pi}}(x \uplus a)$:

$$\begin{aligned} \hat{Q}_{\tilde{\pi}}(x) &= \\ r(x) + \gamma\tilde{\pi}(x,a_v)v(x) &+ \\ \gamma \sum_{a\in\mathcal{A}} \tilde{\pi}(x,a)\Lambda_{\tilde{\pi}}(x \uplus a, \mathcal{T}_{x\uplus a}, \gamma)^T \left(r(\mathcal{T}_{x\uplus a}) + \gamma\tilde{\pi}(\mathcal{T}_{x\uplus a}, a_v) \odot v(\mathcal{T}_{x\uplus a})\right) &= \\ r(x) + \gamma\tilde{\pi}(x,a_v)v(x) + \gamma \sum_{a\in\mathcal{A}} \tilde{\pi}(x,a) \sum_{y\in\mathcal{T}_{x\uplus a}} \Lambda_{\tilde{\pi}}(x\uplus a, y, \gamma)\left(r(y) + \gamma\tilde{\pi}(y,a_v)v(y)\right) &= \\ r(x) + \gamma\tilde{\pi}(x,a_v)v(x) + \sum_{a\in\mathcal{A}} \sum_{y\in\mathcal{T}_{x\uplus a}} \underbrace{\gamma\tilde{\pi}(x,a)\Lambda_{\tilde{\pi}}(x\uplus a, y, \gamma)}_{\Lambda_{\tilde{\pi}}(x,y,\gamma)} \left(r(y) + \gamma\tilde{\pi}(y,a_v)v(y)\right) &= \\ r(x) + \gamma\tilde{\pi}(x,a_v)v(x) + \sum_{a\in\mathcal{A}} \sum_{y\in\mathcal{T}_{x\uplus a}} \Lambda_{\tilde{\pi}}(x,y,\gamma)\left(r(y) + \gamma\tilde{\pi}(y,a_v)v(y)\right)& \end{aligned}$$

Now, note that the terms in the sum is a partition of $\mathcal{T}_x' = \mathcal{T}_x \setminus x$:

$$\cup_{a\in\mathcal{A}} \cup_{y\in\mathcal{T}_{x\uplus a}} \{y\} = \mathcal{T}_x'.$$

We can therefore rewrite the expression as

$$\begin{aligned} \hat{Q}_{\tilde{\pi}}(x) &= \\ r(x) + \gamma\tilde{\pi}(x,a_v)v(x) + \sum_{y\in\mathcal{T}_{x'}} \Lambda_{\tilde{\pi}}(x,y,\gamma)\left(r(y) + \gamma\tilde{\pi}(y,a_v)v(y)\right) &= \\ \sum_{y\in\mathcal{T}_x} \Lambda_{\tilde{\pi}}(x,y,\gamma)\left(r(y) + \gamma\tilde{\pi}(y,a_v)v(y)\right) &= \\ \Lambda_{\tilde{\pi}}(x,\mathcal{T}_x,\gamma)^T r(\mathcal{T}_x) + \Lambda_{\tilde{\pi}}(x, \mathcal{T}_x \uplus a_v, \gamma)^T v(\mathcal{T}_x)& \end{aligned}$$

where we used $\Lambda_{\tilde{\pi}}(x,x,\gamma) = 1$ to rewrite

$$r(x) + \gamma\tilde{\pi}(x,a_v)v(x) = \Lambda_{\tilde{\pi}}(x,x,\gamma)\left(r(x) + \gamma\tilde{\pi}(x,a_v)v(x)\right).$$

This shows that if equation (3.3) holds for all nodes at level L, it also holds for all nodes at level $L-1$. By induction, we have therefore shown that (3.3) holds for all nodes in the tree. ■

The value estimate can be expressed as

$$\hat{V}_{\tilde{\pi}}(x) = \tilde{\pi}(x,a_v)v(x) + \Lambda_{\tilde{\pi}}(x,\mathcal{T}_x',\gamma)^T \left(r(\mathcal{T}_x') + \gamma\tilde{\pi}(\mathcal{T}_x',a_v) \odot v(\mathcal{T}_x')\right)$$

or

$$\hat{V}_{\tilde{\pi}}(x) = \Lambda_{\tilde{\pi}}(x,\mathcal{T}_x',\gamma)^T r(\mathcal{T}_x') + \Lambda_{\tilde{\pi}}(x,\mathcal{T}_x \uplus a_v,\gamma)^T v(\mathcal{T}_x)$$

where $\mathcal{T}_x' = \mathcal{T}_x \setminus x$. Proving the expressions for the value estimates is trivial by applying (3.2) since we know the expanded Q-values.

3.4 The Covariance Matrix

As established in the previous section, the variance of the value estimate is given by

$$\mathbb{V}[\hat{V}_{\tilde{\pi}}(x)] = \tilde{\pi}(x, \mathcal{A}_v)^T \mathrm{Cov}[\hat{Q}_{\tilde{\pi}}(x \uplus \mathcal{A}_v)]\tilde{\pi}(x, \mathcal{A}_v).$$

To evaluate this expression, we need to investigate the covariance matrix further. We will derive a new expression for the variance of the value evaluation by first investigating the variance of the Q-values, the connection between the two will be evident later. We know that

$$\mathbb{V}[\hat{Q}_{\tilde{\pi}}(x)] = \mathbb{V}[r(x) + \gamma \hat{V}_{\tilde{\pi}}(x)] = \mathbb{V}[\Lambda_{\tilde{\pi}}(x, \mathcal{T}_x, \gamma)^T \left(r(\mathcal{T}_x) + \gamma \tilde{\pi}(\mathcal{T}_x, a_v) \odot v(\mathcal{T}_x)\right)].$$

The vector $\Lambda_{\tilde{\pi}}(x, \mathcal{T}_x, \gamma)$ is not a random variable so

$$\mathbb{V}[\hat{Q}_{\tilde{\pi}}(x)] = \Lambda_{\tilde{\pi}}(x, \mathcal{T}_x, \gamma)^T \mathrm{Cov}\left[r(\mathcal{T}_x) + \gamma \tilde{\pi}(\mathcal{T}_x, a_v) \odot v(\mathcal{T}_x)\right] \Lambda_{\tilde{\pi}}(x, \mathcal{T}_x, \gamma)$$

which is an expression for how the variance depends on the covariance between rewards and value evaluations in the tree. In deterministic environments, which is a common assumption for AlphaZero, the variance simplifies to

$$\mathbb{V}[\hat{Q}_{\tilde{\pi}}(x)] = \Lambda_{\tilde{\pi}}(x, \mathcal{T}_x \uplus a_v, \gamma)^\top \mathrm{Cov}[v(\mathcal{T}_x)] \Lambda_{\tilde{\pi}}(x, \mathcal{T}_x \uplus a_v, \gamma).$$

This will be derived in the following section.

3.4.1 Possible Variance Assumtions

To simplify the expression further, we will investigate several optional assumptions on the distributions of r and v to better understand how these assumptions affect the variance of the value estimate.

Reward and Simulation Value Independence

If we assume that the observed rewards and the simulation values are independent then the expression simplifies to

$$\mathbb{V}[\hat{Q}_{\tilde{\pi}}(x)] = \Lambda_{\tilde{\pi}}(x, \mathcal{T}_x, \gamma)^T \mathrm{Cov}\left[r(\mathcal{T}_x)\right] \Lambda_{\tilde{\pi}}(x, \mathcal{T}_x, \gamma) + \Lambda_{\tilde{\pi}}(x, \mathcal{T}_x \uplus a_v, \gamma)^T \mathrm{Cov}\left[v(\mathcal{T}_x)\right] \Lambda_{\tilde{\pi}}(x, \mathcal{T}_x \uplus a_v, \gamma).$$

It is also reasonable to assume that the rewards are independent, that is, the covariance matrix of r is diagonal. Then we can substitute

$$\mathrm{Cov}\left[r(\mathcal{T}_x)\right] = \mathrm{diag}\left(\mathbb{V}[r(\mathcal{T}_x)]\right)$$

where diag is a function mapping a vector to a diagonal matrix. We can also derive a new expression for the variance of the value estimator

$$\mathbb{V}[\hat{V}_{\tilde{\pi}}(x)] = \gamma^{-2}\, \mathbb{V}[\hat{Q}_{\tilde{\pi}}(x)] - \gamma^{-2}\, \mathbb{V}[r(x)].$$

Deterministic Environment

If we operate in a deterministic environment, the reward variance is zero. This simplifies the expressions further to

$$\mathbb{V}[\hat{V}_{\tilde{\pi}}(x)] = \gamma^{-2}\,\mathbb{V}[\hat{Q}_{\tilde{\pi}}(x)] = \gamma^{-2}\Lambda_{\tilde{\pi}}(x, \mathcal{T}_x \uplus a_v, \gamma)^T \mathrm{Cov}\,[v(\mathcal{T}_x)]\,\Lambda_{\tilde{\pi}}(x, \mathcal{T}_x \uplus a_v, \gamma).$$

Independent Simulation Values

Under the assumption that the simulation values are independent, the covariance matrix of the simulation values is diagonal and the variance of the value estimate simplifies to

$$\mathbb{V}[\hat{V}_{\tilde{\pi}}(x)] = \gamma^{-2}\Lambda_{\tilde{\pi}}(x, \mathcal{T}_x \uplus a_v, \gamma)^T \mathrm{diag}\,(\mathbb{V}[v(\mathcal{T}_x)])\,\Lambda_{\tilde{\pi}}(x, \mathcal{T}_x \uplus a_v, \gamma)$$

which is a sum of the variance of the simulation values weighted by the discounted probability of reaching the node

$$\mathbb{V}[\hat{V}_{\tilde{\pi}}(x)] = \sum_{y \in \mathcal{T}_x} \Lambda_{\tilde{\pi}}(x, y, \gamma)^2 \tilde{\pi}(y, a_v)^2\,\mathbb{V}[v(y)].$$

Fixed Variance Simulation Values

If we assume that the variance of each simulation value is σ^2 then the variance of the value estimate simplifies to

$$\mathbb{V}[\hat{V}_{\tilde{\pi}}(x)] = \sigma^2 \sum_{y \in \mathcal{T}_x} \Lambda_{\tilde{\pi}}(x, y, \gamma)^2 \tilde{\pi}(y, a_v)^2 \tag{3.4}$$

and

$$\mathbb{V}[\hat{Q}_{\tilde{\pi}}(x)] = \sigma^2 \Lambda_{\tilde{\pi}}(x, \mathcal{T}_x \uplus a_v, \gamma)^T \Lambda_{\tilde{\pi}}(x, \mathcal{T}_x \uplus a_v, \gamma).$$

3.4.2 The Minimal Variance Evaluator

We now have a better understanding of the variance of the value estimate. In general, an estimator with low variance is desirable. One important tree evaluator is the minimal variance evaluator which for a tree node x is defined as

$$\tilde{\pi}_{\mathrm{Var}} = \arg\min_{\tilde{\pi}} \mathbb{V}[\hat{V}_{\tilde{\pi}}].$$

The solution to this optimization problem is

$$\tilde{\pi}_{\mathrm{Var}}(x, \mathcal{A}_v) = \frac{\mathrm{Cov}[\hat{Q}_{\tilde{\pi}}(x \uplus \mathcal{A}_v)]^{-1}\mathbb{1}}{\mathbb{1}^T \mathrm{Cov}[\hat{Q}_{\tilde{\pi}}(x \uplus \mathcal{A}_v)]^{-1}\mathbb{1}} \propto \mathrm{Cov}[\hat{Q}_{\tilde{\pi}}(x \uplus \mathcal{A}_v)]^{-1}\mathbb{1} \tag{3.5}$$

Proof (3.5). We shorten $\hat{Q} = \hat{Q}_{\tilde{\pi}}(x \uplus \mathcal{A}_v)$, and $\tilde{\pi} = \tilde{\pi}(x, \mathcal{A}_v)$ for breivty

$$\tilde{\pi}_{\text{Var}} = \arg\min_{\tilde{\pi}} \mathbb{V}[\hat{V}_{\tilde{\pi}}(x)] = \arg\min_{\tilde{\pi}} \tilde{\pi}^T \text{Cov}[\hat{Q}]\tilde{\pi}$$

subject to the constraint that $\mathbb{1}^T\tilde{\pi} = 1$ and $\tilde{\pi}(x,a) \geq 0$. We can use the Lagrange multiplier method to solve this optimization problem. The Lagrangian is

$$\mathcal{L}(\tilde{\pi}, \lambda) = \tilde{\pi}^T \text{Cov}[\hat{Q}]\tilde{\pi} + \lambda(\mathbb{1}^T\tilde{\pi} - 1).$$

The gradient with respect to $\tilde{\pi}$ is

$$\nabla_{\tilde{\pi}} \mathcal{L}(\tilde{\pi}, \lambda) = 2\text{Cov}[\hat{Q}]\tilde{\pi} + \lambda\mathbb{1}$$

where $\mathbb{1}$ is a vector of ones and we used that the covariance matrix is symmetric. Note that the gradient is only concerning $\tilde{\pi}(x)$, not $\tilde{\pi}$ for all nodes, this is why we can treat $\hat{Q}$ as a constant. The partial derivative with respect to λ is

$$\frac{\partial \mathcal{L}(\tilde{\pi}, \lambda)}{\partial \lambda} = \mathbb{1}^T\tilde{\pi} - 1.$$

Setting these to zero yields

$$\text{Cov}[\hat{Q}]\tilde{\pi} = -\frac{\lambda}{2}\mathbb{1} \implies \tilde{\pi} = -\frac{\lambda}{2}\text{Cov}[\hat{Q}]^{-1}\mathbb{1}.$$

We know that the inverse exists since a full-rank covariance matrix is symmetric and positive definite. We see that $-\frac{\lambda}{2}$ acts as the normalizer making the sum of the tree evaluation policy equal to one. This yields the final solution

$$\tilde{\pi}_{\text{Var}}(x, \mathcal{A}_v) = \frac{\text{Cov}[\hat{Q}]^{-1}\mathbb{1}}{\mathbb{1}^T\text{Cov}[\hat{Q}]^{-1}\mathbb{1}}.$$

This solution fulfills the constraint $\tilde{\pi}(x,a) \geq 0$ since the inverse of the covariance matrix is positive definite. Further, this solution is a global minimum since the objective function and constraint set are convex. ■

In practice, the covariance matrix would be derived from the covariance of the rewards and simulation values. The solution can be simplified further under independence assumptions where the covariance matrix is a diagonal matrix of variances:

$$\tilde{\pi}_{\text{Var}}(x, a) = \frac{\mathbb{V}[\hat{Q}_{\tilde{\pi}}(x \uplus a)]^{-1}}{\sum_{b \in \mathcal{A}_v} \mathbb{V}[\hat{Q}_{\tilde{\pi}}(x \uplus b)]^{-1}} \propto \mathbb{V}[\hat{Q}_{\tilde{\pi}}(x \uplus a)]^{-1}. \tag{3.6}$$

Proving this is trivial by substituting the covariance matrix with the diagonal matrix in the previous proof.

3.5 The Visitation Count Evaluator

Node visitation counts play an important role in many previous works on MCTS [16]. We define the visitation count $N(x)$ of a node x as

$$N(x) = |\mathcal{T}_x|$$

or equivalently

$$N(x) = 1 + \sum_{a \in \mathcal{A}} N(x \uplus a) = \sum_{a \in \mathcal{A}_v} N(x \uplus a)$$

for all expanded nodes if we consider the visitation count of $N(x \uplus a_v) = 1$. The visitation count of unexpanded nodes is zero. In classical MCTS, the visitation count for terminal nodes can be greater than one even if it does not have any children [44] since it is simply incremented by the backup. To keep the definitions consistent, we let $N(x \uplus a_v) \geq 1$ for terminal nodes. Under the perspective of tree evaluation, the default tree evaluation policy in AlphaZero [42] is the visitation count evaluator defined as

$$\tilde{\pi}_N(x,a) = \frac{N(x \uplus a)}{N(x)} \propto N(x \uplus a).$$

This policy is used for final action selection as well as for estimating the Q-values of nodes in the selection policy. This evaluator seems somewhat arbitrary but we will highlight three important properties making it powerful.

Trajectory Averaging

The first important property of the visitation count evaluator is that

$$\hat{Q}_{\tilde{\pi}_N} = \bar{Q} \tag{3.7}$$

that is, if we evaluate a decision node with the visitation count evaluator, the estimate is the arithmetic average return of trajectories passing through the node. This stems from the property that under the visitation count policy, the probability of reaching each node is proportional to its visits and each value estimate has an equal probability

$$\Lambda_{\tilde{\pi}_N}(x,y,1) = \frac{N(y)}{N(x)},$$
$$\Lambda_{\tilde{\pi}_N}(x,y,1)\tilde{\pi}_N(y,a_v) = \frac{1}{N(x)}.$$

For example, with a discount factor of 1, we can express both the average and the visitation count estimate as

$$\hat{Q}_{\tilde{\pi}_N}(x) = \frac{\mathbb{1}^T}{N(x)} \left(N(\mathcal{T}_x) \odot r(\mathcal{T}_x) + v(\mathcal{T}_x)\right) = \frac{1}{N(x)} \sum_{y \in \mathcal{T}_x} \left(N(y)r(y) + v(y)\right) = \bar{Q}(x).$$

where $\mathbb{1}$ is a vector of ones.

Proof (3.7). We utilize equation (3.3):

$$\begin{aligned}\hat{Q}_{\tilde{\pi}_N}(x) &= \\ \Lambda_{\tilde{\pi}_N}(x, \mathcal{T}_x, \gamma)^T \left(r(\mathcal{T}_x) + \gamma\tilde{\pi}_N(\mathcal{T}_x, a_v) \odot v(\mathcal{T}_x)\right) &= \\ \sum_{y \in \mathcal{T}_x} \Lambda_{\tilde{\pi}_N}(x, y, \gamma)\left(r(y) + \gamma\tilde{\pi}_N(y, a_v) v(y)\right).\end{aligned}$$

We start by investigating $\Lambda_{\tilde{\pi}_N}(x, y, \gamma)$. Let $l(y) = l_x(y) = |\text{path}(x, y)|$ denote the depth level of node y in the tree (of node x by default). Then

$$\begin{aligned}\Lambda_{\tilde{\pi}_N}(x, y, \gamma) &= \\ \prod_{(z,a) \in \text{path}(x,y)} \gamma\tilde{\pi}_N(z, a) &= \\ \gamma^{l(y)} \prod_{(z,a) \in \text{path}(x,y)} \frac{N(z \uplus a)}{N(z)} &= \\ \gamma^{l(y)} \frac{N(y)}{N(x)} \prod_{(z,a) \in \text{path}(x,y)} \frac{N(z)}{N(z)} &= \\ \gamma^{l(y)} \frac{N(y)}{N(x)}\end{aligned}$$

where we used that $N(z \uplus a)$ is simply a shift in the path forward which we can also express by multiplying by the last factor $N(y)$ and dividing by the first factor $N(x)$. Substituting this into the expression for the Q-value estimate yields

$$\begin{aligned}\hat{Q}_{\tilde{\pi}_N}(x) &= \\ \sum_{y \in \mathcal{T}_x} \gamma^{l(y)} \frac{N(y)}{N(x)} \left(r(y) + \gamma \frac{1}{N(y)} v(y)\right) &= \\ \frac{1}{N(x)} \sum_{y \in \mathcal{T}_x} \gamma^{l(y)} \left(N(y) r(y) + \gamma v(y)\right) &= \\ \frac{1}{|T_x|} \sum_{y \in \mathcal{T}_x} |T_y| \gamma^{l(y)} r(y) + \gamma^{l(y)+1} v(y) &= \\ \underbrace{\frac{1}{|T_x|} \sum_{y \in \mathcal{T}_x}}_{\text{Average}} \underbrace{\sum_{(z,_) \in \text{path}(x,y)} \gamma^{l(z)} r(z) + \gamma^{l(y)+1} v(y)}_{\text{Discounted return}} &= \\ \bar{Q}(x).\end{aligned}$$

■

Mimics Selection Policy

The second important property of the visitation count policy is that if the decision tree is built like in MCTS/AlphaZero, by sampling from a selection policy at each node to determine which node to expand, then the visitation count policy will be a reflection of the selection policy. A simple example of this property is when the selection policy is uniform, then the visitation counts will also be uniformly distributed. The value of the visitation count policy will also converge to the value of the selection policy. We will start by showing that the visitation count policy will almost surely converge to the selection policy

$$\lim_{N(x)\to\infty} \pi_N(x) \overset{a.s.}{=} \pi_{\text{Select}}(x). \tag{3.8}$$

Proof (3.8). This can be proven with the law of large numbers. For simplicity, we assume that π_{Select} is static (not the case for UCT). Let $Y_k(x,a)$ be a Bernoulli random variable indicating if action a was selected at node x in the k-th iteration. Then the visitation count policy can be expressed as

$$\pi_N(x,a) = \frac{N(x \cup a)}{N(x)} = \frac{1}{N(x)} \sum_{k=1}^{N(x)} Y_k(x,a).$$

The mean of $Y_k(x,a)$ is $\pi_{\text{Select}}(x,a)$ which lets us apply the law of large numbers:

$$\pi_N(x,a) = \frac{1}{N(x)} \sum_{k=1}^{N(x)} Y_k(x,a) \xrightarrow{a.s.} \pi_{\text{Select}}(x,a) \quad \text{as} \quad N(x) \to \infty.$$

■

In practice, we, for instance, use UCT/PUCT as the selection policy [29, 38]. Both of these policies will change as the tree grows. In the limit of an infinite number of visits, the exploration term will converge to zero and they will simply follow the Q-value policy

$$\lim_{N(x)\to\infty} \pi_{\text{UCT}}(x) = \lim_{N(x)\to\infty} \pi_{\text{PUCT}}(x) = \pi_Q(x). \tag{3.9}$$

Proof (3.9). The exploration term will converge to zero for both UCT and PUCT. The exploration term for UCT is

$$U^{\text{UCT}}(x,a) = c \cdot \sqrt{\frac{\log N(x)}{N(x \uplus a)}}$$

and for PUCT it is

$$U^{\text{PUCT}}(x,a) = c \cdot \pi_\theta(x,a) \cdot \frac{\sqrt{N(x)}}{1 + N(x \uplus a)}.$$

We know that $N(x \uplus a) \leq N(x)\ \forall a \in \mathcal{A}_v$ and $U^{\text{UCT}} \geq 0$ so

$$0 \leq \lim_{N(x)\to\infty} U^{\text{UCT}}(x,a) \leq \lim_{N(x)\to\infty} c \cdot \sqrt{\frac{\log N(x)}{N(x)}} = 0.$$

Similarly, for PUCT we have

$$0 \leq \lim_{N(x)\to\infty} U^{\text{PUCT}}(x,a) \leq \lim_{N(x)\to\infty} c \cdot \pi_\theta(x,a) \cdot \frac{\sqrt{N(x)}}{1+N(x)} = 0.$$

These limits are trivial since logarithm and square root grow sub-linearly. There are some additional technicalities to consider for this proof. Please consult literature on multi-armed bandits for all the details [2]. ■

With default UCT and PUCT, the Q-values used in the selection policy are the average returns $\bar{Q}$ of the nodes which, as we showed in the previous section, is equivalent to $\hat{Q}_{\tilde{\pi}_N}$ which will converge to $\hat{Q}_{\tilde{\pi}_Q}$ with enough visits. Conceptually, this implies that

$$\lim_{N(x)\to\infty} \pi_N(x) = \lim_{N(x)\to\infty} \pi_{\text{Select}}(x) = \underbrace{\pi_Q(x)}_{\text{If UCT / PUCT}}$$

so

$$\lim_{N(x)\to\infty} \hat{V}_{\tilde{\pi}_N}(x) = \lim_{N(x)\to\infty} \hat{V}_{\pi_{\text{Select}}}(x) = \underbrace{\lim_{N(x)\to\infty} \hat{V}_{\tilde{\pi}_Q}(x)}_{\text{If UCT / PUCT}} = V^*(x).$$

This means that the combination of a selection policy like UCT or PUCT and the visitation count evaluator should converge to the optimal value for the root node with enough visits. This is an important theoretical result since it suggests that with enough simulation budget, the visitation count evaluator will converge to an optimal policy [29].

Variance Minimizing

The final interesting property of the visitation count evaluator is that it minimizes the variance of the value estimate under certain assumptions. In a deterministic environment with fixed variance simulation values and discount $\gamma = 1$

$$\tilde{\pi}_N = \tilde{\pi}_{\text{Var}}. \tag{3.10}$$

Proof (3.10). Under the independence assumption, we know from equation (3.6) that the minimal variance evaluator is given by

$$\tilde{\pi}_{\text{Var}}(x,a) \propto \mathbb{V}[\hat{Q}_{\tilde{\pi}}(x \uplus a)]^{-1}.$$

The question that remains is, what is the variance of each child node $\mathbb{V}[\hat{Q}_{\tilde{\pi}}(x \uplus a)]$?

We postulate that with fixed variance, the Q-variance when using the minimal variance evaluator is

$$\mathbb{V}[\hat{Q}_{\tilde{\pi}}(x)] = \mathbb{V}[\hat{Q}_{\tilde{\pi}_{\text{Var}}}(x)] = \mathbb{V}[\hat{Q}_{\tilde{\pi}_N}(x)] = \frac{\sigma^2}{N(x)}$$

which we will prove by induction. Preserve the notation in proof (3.3).

Base case: $l(x) = n$. For leaf nodes, the variance the minimal variance evaluator is

$$\mathbb{V}[\hat{Q}_{\tilde{\pi}_{\text{Var}}}(x)] = \mathbb{V}[r(x) + \gamma v(x)] = \underbrace{0 + \sigma^2}_{\text{Fixed variance}} = \frac{\sigma^2}{1} = \frac{\sigma^2}{N(x)}.$$

Induction step: $l(x) = L - 1$. Assume that the variance of the Q-value estimate for all nodes at level $L-1$ is $\sigma^2/N(x)$. Under the assumption that $\gamma = 1$ and simulation value independence, the variance of the Q-value estimate for all nodes at level L is

$$\begin{aligned}
\mathbb{V}[\hat{Q}_{\tilde{\pi}_{\text{Var}}}(x)] &= \\
\mathbb{V}[\hat{V}_{\tilde{\pi}_{\text{Var}}}(x)] &= \\
\sum_{a \in \mathcal{A}_v} \tilde{\pi}_{\text{Var}}(x,a)^2 \mathbb{V}[\hat{Q}_{\tilde{\pi}_{\text{Var}}}(x \uplus a)] &= \\
\sum_{a \in \mathcal{A}_v} \left(\frac{\mathbb{V}[\hat{Q}_{\tilde{\pi}_{\text{Var}}}(x \uplus a)]^{-1}}{\sum_{b \in \mathcal{A}_v} \mathbb{V}[\hat{Q}_{\tilde{\pi}_{\text{Var}}}(x \uplus b)]^{-1}} \right)^2 \mathbb{V}[\hat{Q}_{\tilde{\pi}_{\text{Var}}}(x \uplus a)] &= \\
\sum_{a \in \mathcal{A}_v} \frac{\mathbb{V}[\hat{Q}_{\tilde{\pi}_{\text{Var}}}(x \uplus a)]^{-1}}{\left(\sum_{b \in \mathcal{A}_v} \mathbb{V}[\hat{Q}_{\tilde{\pi}_{\text{Var}}}(x \uplus b)]^{-1}\right)^2} &= \\
\left(\sum_{a \in \mathcal{A}_v} \mathbb{V}[\hat{Q}_{\tilde{\pi}_{\text{Var}}}(x \uplus a)]^{-1} \right)^{-2} \left(\sum_{a \in \mathcal{A}_v} \mathbb{V}[\hat{Q}_{\tilde{\pi}_{\text{Var}}}(x \uplus a)]^{-1} \right) &= \\
\left(\sum_{a \in \mathcal{A}_v} \mathbb{V}[\hat{Q}_{\tilde{\pi}_{\text{Var}}}(x \uplus a)]^{-1} \right)^{-1} &.
\end{aligned}$$

Now, all nodes $x \uplus a$ are at level $L-1$ and we can apply the induction hypothesis to get

$$\begin{aligned}
\left(\sum_{a \in \mathcal{A}_v} \left(\frac{\sigma^2}{N(x \uplus a)} \right)^{-1} \right)^{-1} &= \\
\left(\sum_{a \in \mathcal{A}_v} \frac{N(x \uplus a)}{\sigma^2} \right)^{-1} &= \\
\frac{\sigma^2}{\sum_{a \in \mathcal{A}_v} N(x \uplus a)} &= \\
\frac{\sigma^2}{N(x)} &.
\end{aligned}$$

This completes the proof of the variance of the minimal variance evaluator in this setting. Finally, we can use that

$$\tilde{\pi}_{\mathrm{Var}}(x,a) \propto \mathbb{V}[\hat{Q}_{\tilde{\pi}}(x \uplus a)]^{-1} = \left(\frac{\sigma^2}{N(x \uplus a)}\right)^{-1} = \frac{N(x \uplus a)}{\sigma^2} \propto \frac{N(x \uplus a)}{N(x)} = \tilde{\pi}_N(x,a)$$

to show that the visitation count and the minimal variance evaluator are equivalent in this setting. The tree evaluators being proportional means they are equal since they are probability distributions. ■

The variance of the visitation count evaluator is given by

$$\mathbb{V}[\hat{V}_{\tilde{\pi}_N}(x)] = \frac{1}{N(x)^2}\mathbb{1}^T \mathrm{Cov}\left[v(\mathcal{T}_x)\right]\mathbb{1} = \underbrace{\frac{1}{N(x)^2}\mathbb{1}^T\,\mathbb{V}[v(\mathcal{T}_x)] =}_{\text{If independent}} \underbrace{\frac{\sigma^2}{N(x)}}_{\text{If fixed variance}} \propto \frac{1}{N(x)}.$$

This is an important result since it forms a connection between the variance and the visitation counts.

3.6 General Tree Evaluation

Up to this point, we have concentrated on the Q-evaluator $\tilde{\pi}_Q$ and the visitation evaluator $\tilde{\pi}_N$. The former yields a high value but with high variance, whereas the latter offers low variance but may underestimate the optimal value. Additionally, the visitation evaluator is inherently dependent on how the tree is constructed. In this section, we will investigate alternative, novel, evaluators and try to find a middle ground between $\tilde{\pi}_Q$ and $\tilde{\pi}_N$.

Chebyshev Evaluator

One way of approaching this is to consider $\hat{V}_{\tilde{\pi}}$ as a distribution parameterized by the evaluator $\tilde{\pi}$. Previously, we found the distribution that maximized its observed value or minimized the variance. To combine these, we will instead attempt to maximize a probabilistic lower bound on its expected value. Consider the optimization

$$\begin{aligned} \arg\max_{\tilde{\pi}} \quad & B_{\tilde{\pi}} \\ \text{s.t.} \quad & \mathbb{P}\left(\mathbb{E}[\hat{V}_{\tilde{\pi}}] \geq B_{\tilde{\pi}}\right) \geq 1-\delta \end{aligned}$$

for $\delta \in (0,1]$. That is, we find an evaluator that is good with high probability. Depending on what we know about the distribution of $\hat{V}_{\tilde{\pi}}$ we can solve this in different ways. We have already investigated the mean and variance of this distribution and could use these to construct a lower bound. Chebyshev's inequality states that, for any $k \in \mathbb{R}^+$

$$\frac{1}{k^2} \geq \mathbb{P}\left(|\hat{V}_{\tilde{\pi}} - \mathbb{E}[\hat{V}_{\tilde{\pi}}]| \geq k\sqrt{\mathbb{V}[\hat{V}_{\tilde{\pi}}]}\right) \geq \mathbb{P}\left(\hat{V}_{\tilde{\pi}} - \mathbb{E}[\hat{V}_{\tilde{\pi}}] \geq k\sqrt{\mathbb{V}[\hat{V}_{\tilde{\pi}}]}\right) = \mathbb{P}\left(\hat{V}_{\tilde{\pi}} - k\sqrt{\mathbb{V}[\hat{V}_{\tilde{\pi}}]} \geq \mathbb{E}[\hat{V}_{\tilde{\pi}}]\right)$$

so

$$\mathbb{P}\left(\mathbb{E}[\hat{V}_{\tilde{\pi}}] \geq \hat{V}_{\tilde{\pi}} - k\sqrt{\mathbb{V}[\hat{V}_{\tilde{\pi}}]}\right) \geq 1 - \frac{1}{k^2}$$

which means that

$$\tilde{\pi}_{\text{Chebyshev}} = \arg\max_{\tilde{\pi}} \hat{V}_{\tilde{\pi}} - \frac{1}{\sqrt{\delta}}\sqrt{\mathbb{V}[\hat{V}_{\tilde{\pi}}]}$$

provide one feasible solution to the inequality under the current distributional assumptions. It might be possible to construct tighter bounds by introducing additional assumptions such as bounded values (see Chernoff, Hoeffding, Bernstein) [12, 5, 24]. Note that

$$\lim_{\delta \to 0^+} \tilde{\pi}_{\text{Chebyshev}} = \tilde{\pi}_{\text{Var}}$$

and

$$\lim_{\delta \to \infty} \tilde{\pi}_{\text{Chebyshev}} = \tilde{\pi}_{Q}.$$

Finding an analytical solution for the Chebyshev evaluator is difficult but if we relax the lower bound further and assume value estimate independence we find the solution

$$\tilde{\pi}_{\text{Chebyshev}'} = \operatorname*{PolicyMax}_{\mathcal{A}_v}\left[\hat{Q}_{\tilde{\pi}}(x \uplus a) - \frac{1}{\sqrt{\delta}}\sqrt{\mathbb{V}[\hat{Q}_{\tilde{\pi}}(x \uplus a)]}\right] \tag{3.11}$$

which is a deterministic policy (assuming no ties).

Proof (3.11). First, note that under the independence assumption, we can find an upper bound on the standard deviation of the value estimate

$$\sqrt{\mathbb{V}[\hat{V}_{\tilde{\pi}}]} = \sqrt{\sum_{a \in \mathcal{A}_v} \tilde{\pi}(x,a)^2\, \mathbb{V}[\hat{Q}_{\tilde{\pi}}(x \uplus a)]} \leq \sum_{a \in \mathcal{A}_v} \tilde{\pi}(x,a)\sqrt{\mathbb{V}[\hat{Q}_{\tilde{\pi}}(x \uplus a)]}$$

where we used the Cauchy–Schwarz (triangle) inequality. This implies

$$\begin{aligned} \hat{V}_{\tilde{\pi}} - k\sqrt{\mathbb{V}[\hat{V}_{\tilde{\pi}}]} \geq \\ \hat{V}_{\tilde{\pi}} - k\sum_{a \in \mathcal{A}_v} \tilde{\pi}(x,a)\sqrt{\mathbb{V}[\hat{Q}_{\tilde{\pi}}(x \uplus a)]} = \\ \sum_{a \in \mathcal{A}_v} \tilde{\pi}(x,a)\hat{Q}_{\tilde{\pi}}(x \uplus a) - k\sum_{a \in \mathcal{A}_v} \tilde{\pi}(x,a)\sqrt{\mathbb{V}[\hat{Q}_{\tilde{\pi}}(x \uplus a)]} = \\ \sum_{a \in \mathcal{A}_v} \tilde{\pi}(x,a)\left(\hat{Q}_{\tilde{\pi}}(x \uplus a) - k\sqrt{\mathbb{V}[\hat{Q}_{\tilde{\pi}}(x \uplus a)]}\right). \end{aligned}$$

We can therefore conclude that

$$\mathbb{P}\left(\mathbb{E}[\hat{V}_{\tilde{\pi}}] \geq \sum_{a \in \mathcal{A}_v} \tilde{\pi}(x,a)\left(\hat{Q}_{\tilde{\pi}}(x \uplus a) - k\sqrt{\mathbb{V}[\hat{Q}_{\tilde{\pi}}(x \uplus a)]}\right)\right) \geq 1 - \frac{1}{k^2}$$

which means that we have another feasible lower bound from

$$\tilde{\pi}_{\text{Chebyshev}'} = \arg\max_{\tilde{\pi}} \sum_{a \in \mathcal{A}_v} \tilde{\pi}(x,a) \left(\hat{Q}_{\tilde{\pi}}(x \uplus a) - \frac{1}{\sqrt{\delta}} \sqrt{\mathbb{V}[\hat{Q}_{\tilde{\pi}}(x \uplus a)]} \right).$$

This trivially has the analytical solution

$$\tilde{\pi}_{\text{Chebyshev}'} = \underset{\mathcal{A}_v}{\text{PolicyMax}} \left[\hat{Q}_{\tilde{\pi}}(x \uplus a) - \frac{1}{\sqrt{\delta}} \sqrt{\mathbb{V}[\hat{Q}_{\tilde{\pi}}(x \uplus a)]} \right]. \tag{3.12}$$

■

Mean-Variance Evaluator

The relaxed Chebyshev evaluator is a good start but it returns a deterministic policy which is not ideal in certain scenarios. We investigate the Mean-Variance evaluator where we replace the standard deviation with variance in the optimization problem

$$\tilde{\pi}_{\text{MV}} = \arg\max_{\tilde{\pi}} \hat{V}_{\tilde{\pi}} - \lambda \mathbb{V}[\hat{V}_{\tilde{\pi}}] \text{ with } \lambda \geq 0$$

which has the solution

$$\tilde{\pi}_{\text{MV}}(x,\mathcal{A}_v) = \tilde{\pi}_{\text{Var}}(x,\mathcal{A}_v) + \frac{1}{2\lambda} \text{Cov}\left[\hat{Q}_{\tilde{\pi}}(x \uplus \mathcal{A}_v)\right]^{-1} \left(\hat{Q}_{\tilde{\pi}}(x \uplus \mathcal{A}_v) - \tilde{\pi}_{\text{Var}}(x,\mathcal{A}_v)^T \hat{Q}_{\tilde{\pi}}(x \uplus \mathcal{A}_v) \mathbb{1} \right). \tag{3.13}$$

To avoid negative policy values we have the additional constraint that

$$\lambda \geq \max_{a \in \mathcal{A}_v} \frac{1}{2\tilde{\pi}_{\text{Var}}(x,a)} \left[\text{Cov}\left[\hat{Q}\right]^{-1} \right]_a^T \left(\hat{Q} - \tilde{\pi}_{\text{Var}}^T \hat{Q} \mathbb{1} \right)$$

where $[\Sigma]_a$ denotes a vector corresponding to row a in Σ.

Proof (3.13). We shorten $\hat{Q} = \hat{Q}_{\tilde{\pi}}(x \uplus \mathcal{A}_v)$, and $\tilde{\pi} = \tilde{\pi}(x,\mathcal{A}_v)$ for breivty. The optimization problem can be written as

$$\begin{aligned} \tilde{\pi}_{\text{MV}} &= \\ \arg\max_{\tilde{\pi}} \hat{V}_{\tilde{\pi}} - \lambda \mathbb{V}[\hat{V}_{\tilde{\pi}}] &= \\ \arg\max_{\tilde{\pi}} \tilde{\pi}^T \hat{Q} - \lambda \tilde{\pi}^T \text{Cov}\left[\hat{Q}\right] \tilde{\pi} & \end{aligned}$$

subject to the constraint that $\tilde{\pi}^T \mathbb{1} = 1$ and $\tilde{\pi}(x,a) \geq 0,\ \forall a \in \mathcal{A}_v$. We can use the Lagrange multiplier method to solve this optimization problem. To find the optimal policy

$\tilde{\pi}_{\mathrm{MV}}$, let's set up the Lagrangian to enforce the constraint that $\tilde{\pi}$ is a valid probability distribution:

$$\mathcal{L}(\tilde{\pi},\kappa) = \tilde{\pi}^T\hat{Q} - \lambda\tilde{\pi}^T \mathrm{Cov}\left[\hat{Q}\right]\tilde{\pi} + \kappa\left(\tilde{\pi}^T\mathbb{1} - 1\right).$$

Taking the gradient with respect to $\tilde{\pi}(x)$ yields

$$\nabla_{\tilde{\pi}(x)}\mathcal{L} = \hat{Q} - 2\lambda \mathrm{Cov}\left[\hat{Q}\right]\tilde{\pi} + \kappa\mathbb{1}$$

since $x \notin x \uplus \mathcal{A}_v$. The derivative with respect to κ gives

$$\frac{\partial\mathcal{L}}{\partial\kappa} = \tilde{\pi}^T\mathbb{1} - 1.$$

Setting the derivatives to zero yields

$$\nabla_{\tilde{\pi}(x)}\mathcal{L} = 0 \iff 2\lambda\mathrm{Cov}\left[\hat{Q}\right]\tilde{\pi} = \hat{Q} + \kappa\mathbb{1} \iff \tilde{\pi} = \frac{1}{2\lambda}\mathrm{Cov}\left[\hat{Q}\right]^{-1}\left(\hat{Q} + \kappa\mathbb{1}\right).$$

We can add the constraint $\mathbb{1}^T\tilde{\pi} = 1$ to find κ:

$$\begin{aligned}
\mathbb{1}^T\tilde{\pi} = \mathbb{1}^T\frac{1}{2\lambda}\mathrm{Cov}\left[\hat{Q}\right]^{-1}\left(\hat{Q} + \kappa\mathbb{1}\right) = 1 &\iff \\
\kappa\mathbb{1}^T\mathrm{Cov}\left[\hat{Q}\right]^{-1}\mathbb{1} = 2\lambda - \mathbb{1}^T\mathrm{Cov}\left[\hat{Q}\right]^{-1}\hat{Q} &\iff \\
\kappa = \frac{2\lambda}{\mathbb{1}^T\mathrm{Cov}\left[\hat{Q}\right]^{-1}\mathbb{1}} - \frac{\mathbb{1}^T\mathrm{Cov}\left[\hat{Q}\right]^{-1}}{\mathbb{1}^T\mathrm{Cov}\left[\hat{Q}\right]^{-1}\mathbb{1}}\hat{Q}&
\end{aligned}$$

where we can identify the minimal variance evaluator from (3.5)

$$\frac{\mathbb{1}^T\mathrm{Cov}\left[\hat{Q}\right]^{-1}}{\mathbb{1}^T\mathrm{Cov}\left[\hat{Q}\right]^{-1}\mathbb{1}}\hat{Q} = \underbrace{\left(\frac{\mathrm{Cov}\left[\hat{Q}\right]^{-1}\mathbb{1}}{\mathbb{1}^T\mathrm{Cov}\left[\hat{Q}\right]^{-1}\mathbb{1}}\right)^T}_{\tilde{\pi}_{\mathrm{Var}}^T}\hat{Q}$$

where we used that the inverse of the covariance matrix is symmetric. Substituting this into the policy yields

$$\begin{aligned}
\tilde{\pi} &= \\
\frac{1}{2\lambda}\mathrm{Cov}\left[\hat{Q}\right]^{-1}\left(\hat{Q} + \frac{2\lambda}{\mathbb{1}^T\mathrm{Cov}\left[\hat{Q}\right]^{-1}\mathbb{1}}\mathbb{1} - \tilde{\pi}_{\mathrm{Var}}^T\hat{Q}\mathbb{1}\right) &= \\
\frac{\mathrm{Cov}\left[\hat{Q}\right]^{-1}\mathbb{1}}{\mathbb{1}^T\mathrm{Cov}\left[\hat{Q}\right]^{-1}\mathbb{1}} + \frac{1}{2\lambda}\mathrm{Cov}\left[\hat{Q}\right]^{-1}\left(\hat{Q} - \tilde{\pi}_{\mathrm{Var}}^T\hat{Q}\mathbb{1}\right) &= \\
\tilde{\pi}_{\mathrm{Var}} + \frac{1}{2\lambda}\mathrm{Cov}\left[\hat{Q}\right]^{-1}\left(\hat{Q} - \tilde{\pi}_{\mathrm{Var}}^T\hat{Q}\mathbb{1}\right)&
\end{aligned}$$

This is a global maximum since the objective function is concave and the constraint set is convex. The solution holds as long as all probabilities are non-negative which induces a tighter lower bound on λ.

$$
\begin{aligned}
[\tilde{\pi}_{\mathrm{MV}}]_a = \left[\tilde{\pi}_{\mathrm{Var}} + \frac{1}{2\lambda}\mathrm{Cov}\left[\hat{Q}\right]^{-1}\left(\hat{Q} - \tilde{\pi}_{\mathrm{Var}}^T \hat{Q}\mathbb{1}\right)\right]_a \geq 0, \forall a \in \mathcal{A}_v \iff \\
\frac{1}{2\lambda}\left[\mathrm{Cov}\left[\hat{Q}\right]^{-1}\left(\hat{Q} - \tilde{\pi}_{\mathrm{Var}}^T \hat{Q}\mathbb{1}\right)\right]_a \geq -[\tilde{\pi}_{\mathrm{Var}}]_a, \forall a \in \mathcal{A}_v \iff \\
\lambda \geq \frac{1}{2[\tilde{\pi}_{\mathrm{Var}}]_a}\left[\mathrm{Cov}\left[\hat{Q}\right]^{-1}\left(\hat{Q} - \tilde{\pi}_{\mathrm{Var}}^T \hat{Q}\mathbb{1}\right)\right]_a, \forall a \in \mathcal{A}_v \iff \\
\lambda \geq \max_{a \in \mathcal{A}_v} \frac{1}{2\tilde{\pi}_{\mathrm{Var}}(x,a)}\left[\mathrm{Cov}\left[\hat{Q}\right]^{-1}\right]_a^T \left(\hat{Q} - \tilde{\pi}_{\mathrm{Var}}^T \hat{Q}\mathbb{1}\right)
\end{aligned}
$$

where I used $[Av]_a = [A]_a^T v$ if $[A]$ extracts a row vector from the matrix. All these matrices/vectors can be indexed by a. ■

Note that the evaluator is a perturbation of the minimal variance evaluator where the weights are increased for actions with higher Q-values. Similarly to the Chebyshev evaluator, the Mean-Variance evaluator will converge to the minimal variance evaluator as $\lambda \to \infty$ and to the Q evaluator as $\lambda \to 0^+$:

$$\lim_{\lambda \to 0^+} \tilde{\pi}_{\mathrm{MV}} = \tilde{\pi}_Q$$

and

$$\lim_{\lambda \to \infty} \tilde{\pi}_{\mathrm{MV}} = \tilde{\pi}_{\mathrm{Var}}.$$

The Mean-Variance evaluator does not suffer from the issues of being deterministic but the constraint on λ makes it impractical.

Mean-Variance Constrained Evaluator

The Mean-Variance Constrained (MVC) evaluator maximizes the expected value but constrains the policy to stay close to the minimal variance evaluator. The evaluator is defined as

$$\tilde{\pi}_{\mathrm{MVC}} = \arg\max_{\tilde{\pi}} \hat{V}_{\tilde{\pi}} - \frac{1}{\beta}\mathrm{KL}(\tilde{\pi}, \tilde{\pi}_{\mathrm{Var}}) \text{ with } \beta > 0$$

which has the solution

$$\tilde{\pi}_{\mathrm{MVC}}(x,a) = \frac{\tilde{\pi}_{\mathrm{Var}}(x,a)\exp\left(\beta\hat{Q}_{\tilde{\pi}}(x \uplus a)\right)}{\sum_{b \in \mathcal{A}_v} \tilde{\pi}_{\mathrm{Var}}(x,b)\exp\left(\beta\hat{Q}_{\tilde{\pi}}(x \uplus b)\right)} \propto \tilde{\pi}_{\mathrm{Var}}(x,a)\exp\left(\beta\hat{Q}_{\tilde{\pi}}(x \uplus a)\right). \quad (3.14)$$

Proof (3.14).

$$\begin{aligned}
\tilde{\pi}_{\text{MVC}} &= \\
\arg\max_{\tilde{\pi}} \hat{V}_{\tilde{\pi}} - \frac{1}{\beta}\text{KL}(\tilde{\pi}, \tilde{\pi}_{\text{Var}}) &= \\
\arg\max_{\tilde{\pi}} \sum_{a \in \mathcal{A}_v} \tilde{\pi}(x,a)\hat{Q}_{\tilde{\pi}}(x \uplus a) - \frac{1}{\beta} \sum_{a \in \mathcal{A}_v} \tilde{\pi}(x,a) \log\left(\frac{\tilde{\pi}(x,a)}{\tilde{\pi}_{\text{Var}}(x,a)}\right) &= \\
\arg\max_{\tilde{\pi}} \sum_{a \in \mathcal{A}_v} \tilde{\pi}(x,a)\left(\hat{Q}_{\tilde{\pi}}(x \uplus a) - \frac{1}{\beta}\log\left(\frac{\tilde{\pi}(x,a)}{\tilde{\pi}_{\text{Var}}(x,a)}\right)\right)
\end{aligned}$$

To find the optimal policy $\tilde{\pi}_{\text{MVC}}$, let's set up the Lagrangian to enforce the constraint that $\tilde{\pi}$ is a valid probability distribution:

$$\mathcal{L}(\tilde{\pi}, \lambda) = \sum_{a \in \mathcal{A}_v} \tilde{\pi}(x,a)\left(\hat{Q}_{\tilde{\pi}}(x \uplus a) - \frac{1}{\beta}\log\left(\frac{\tilde{\pi}(x,a)}{\tilde{\pi}_{\text{Var}}(x,a)}\right)\right) + \lambda\left(\sum_{a \in \mathcal{A}_v} \tilde{\pi}(x,a) - 1\right).$$

Taking the derivative with respect to $\tilde{\pi}(x,a)$ yields

$$\frac{\partial \mathcal{L}}{\partial \tilde{\pi}(x,a)} = \hat{Q}_{\tilde{\pi}}(x \uplus a) - \frac{1}{\beta}\log\left(\frac{\tilde{\pi}(x,a)}{\tilde{\pi}_{\text{Var}}(x,a)}\right) + 1 + \lambda$$

since $x \notin x \uplus \mathcal{A}_v$. The derivative with respect to λ gives

$$\frac{\partial \mathcal{L}}{\partial \lambda} = \sum_{a \in \mathcal{A}_v} \tilde{\pi}(x,a) - 1.$$

Setting the derivatives to zero yields

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \tilde{\pi}(x,a)} = 0 &\iff \\
\log\left(\frac{\tilde{\pi}(x,a)}{\tilde{\pi}_{\text{Var}}(x,a)}\right) = \beta\left(\hat{Q}_{\tilde{\pi}}(x \uplus a) + 1 + \lambda\right) &\iff \\
\tilde{\pi}(x,a) = \tilde{\pi}_{\text{Var}}(x,a)\exp\left(\beta\left(\hat{Q}_{\tilde{\pi}}(x \uplus a) + 1 + \lambda\right)\right) &\propto \tilde{\pi}_{\text{Var}}(x,a)\exp\left(\beta\hat{Q}_{\tilde{\pi}}(x \uplus a)\right)
\end{aligned}$$

and

$$\frac{\partial \mathcal{L}}{\partial \lambda} = 0 \iff \sum_{a \in \mathcal{A}_v} \tilde{\pi}(x,a) = 1.$$

Combining these gives

$$\tilde{\pi}_{\text{MVC}}(x,a) = \frac{\tilde{\pi}_{\text{Var}}(x,a)\exp\left(\beta\hat{Q}_{\tilde{\pi}}(x \uplus a)\right)}{\sum_{b \in \mathcal{A}_v} \tilde{\pi}_{\text{Var}}(x,b)\exp\left(\beta\hat{Q}_{\tilde{\pi}}(x \uplus b)\right)}.$$

This is a global maximum since the objective function is concave and the constraint set is convex. ■

This evaluator also balances the trade-off between high value and variance while being easier to use than the Mean-Variance evaluator. When $\beta \to \infty$, the MVC evaluator will converge to the Q-evaluator, and when $\beta \to 0^+$ it will converge to the minimal variance evaluator:

$$\lim_{\beta \to 0^+} \tilde{\pi}_{\text{MVC}} = \tilde{\pi}_{\text{Var}}$$

and

$$\lim_{\beta \to \infty} \tilde{\pi}_{\text{MVC}} = \tilde{\pi}_{Q}.$$

3.7 Modified Construction

In the previous section, we introduced several tree evaluators for general decision trees. The primary application for these evaluators is extracting a final policy from a previously constructed tree. In this section, we will investigate how we could additionally use the new evaluators to improve the tree construction process. We will start by considering how we can improve the UCT/PUCT selection algorithms.

3.7.1 Modified Selection

We focus on selection algorithms in the UCB family such as UCT and PUCT [2, 29, 38]. The general form of these selection policies are

$$\pi_{\text{UCB}} = \underset{\mathcal{A}}{\text{PolicyMax}}\left[\text{UCB}\right]$$

with

$$\text{UCB}(x,a) = \bar{Q}(x \uplus a) + U(x,a) \tag{3.15}$$

where $\bar{Q}$ is the average return of the node and U is an exploration term.

In Section 3.5 we showed that the average return is the value estimate of the visitation count evaluator

$$\bar{Q}(x) = \hat{Q}_{\tilde{\pi}_N}(x).$$

This makes sense under the assumption that we use visitation counts to determine the final policy but if we instead use one of the new evaluators, we propose to use the value estimate of the new evaluator instead. This would modify equation (3.15) to

$$\text{UCB}_{\tilde{\pi}}(x,a) = \hat{Q}_{\tilde{\pi}}(x \uplus a) + U(x,a). \tag{3.16}$$

Note that this implies that

$$\text{UCB}_{\tilde{\pi}_N} = \text{UCB}.$$

Furthermore, we could also investigate the exploration term U for UCT and PUCT. For UCT, the exploration term is given by

$$U^{\text{UCT}}(x,a) = c \cdot \sqrt{\frac{\log N(x)}{N(x \uplus a)}}$$

and for PUCT it is

$$U^{\text{PUCT}}(x,a) = c \cdot \pi_\theta(x,a) \cdot \frac{\sqrt{N(x)}}{1+N(x \uplus a)}.$$

The principle both of these algorithms are based on is upper confidence bounds, that is optimistic exploration. They both base their exploration bonus on visitation count. In the default algorithm, $\bar{Q}(x)$ is determined by averaging over $N(x)$ returns. In this setting, the variance of the average return is inversely proportional to the number of visits which is also what we showed in Section 3.5. This property only holds under certain assumptions and if we use $\bar{Q}$. If we use the introduced value estimate instead, we can replace the visitation count with the estimated variance instead. This would give us the following exploration term for UCT

$$U_{\tilde{\pi}}^{\text{UCT}}(x,a) = c \cdot \sqrt{\frac{\log \mathbb{V}[\hat{Q}_{\tilde{\pi}}(x)]^{-1}}{\mathbb{V}[\hat{Q}_{\tilde{\pi}}(x \uplus a)]^{-1}}} = c \cdot \sqrt{\log \mathbb{V}[\hat{Q}_{\tilde{\pi}}(x)]^{-1}} \sqrt{\mathbb{V}[\hat{Q}_{\tilde{\pi}}(x \uplus a)]}$$

where we assume that $\mathbb{V}[\hat{Q}_{\tilde{\pi}}(x)] \leq 1$ so that the square root of the logarithm is real. This can be achieved in any environment by reward scaling. For PUCT, the exploration term would be

$$U_{\tilde{\pi}}^{\text{PUCT}}(x,a) = c \cdot \pi_\theta(x,a) \cdot \frac{\sqrt{\mathbb{V}[\hat{Q}_{\tilde{\pi}}(x)]^{-1}}}{1+\mathbb{V}[\hat{Q}_{\tilde{\pi}}(x \uplus a)]^{-1}}.$$

These modifications to the already well-established selection algorithms could potentially improve the performance of the tree construction process and better capture the essence of optimistic search.

3.7.2 General Construction

The previous section focused on how we could modify UCT or PUCT to improve tree construction. It is important to acknowledge that the true strength of our proposed general tree evaluators is that they are not inherently dependent on any construction algorithm. This is not the case for the default visitation count evaluator since the visitation distribution is determined by the construction algorithm. The possibilities for modified construction are vaster than the already proposed modifications to UCT/PUCT. We could, for example, consider constructing the tree using breadth-first search or using the sequential halving algorithm used by Danihelka et al. [16].

3.8 Numerical Computation

The novel tree evaluators used for final policy extraction as well as in the modified selection policies require access to the value estimate $\hat{Q}_{\tilde{\pi}}$ and its estimated variance $\mathbb{V}[\hat{Q}_{\tilde{\pi}}]$. This section will focus on the computation of these values in practice. The value estimate is recursively defined as

$$\hat{Q}_{\tilde{\pi}}(x) = r(x) + \gamma \sum_{a \in \mathcal{A}_v} \tilde{\pi}(x,a) \hat{Q}_{\tilde{\pi}}(x \uplus a).$$

In general, we assume that

$$\tilde{\pi}(x,a) = 0 \; \forall \; a \in \mathcal{A} \text{ if } |\mathcal{T}_x| = 1.$$

This implies that the value estimate for leaf nodes is

$$\hat{Q}_{\tilde{\pi}}(x) = r(x) + \gamma \hat{Q}_{\tilde{\pi}}(x \uplus a_v) = r(x) + \gamma v(x)$$

since $\tilde{\pi}$ is a probability distribution. In practice, when computing the value estimate, we therefore start with the leaf nodes and then recursively evaluate the value of the parent nodes until we reach the root. We follow a similar procedure to compute the variance, under the assumption of independent rewards and value estimates, the variance can be computed as

$$\mathbb{V}[\hat{Q}_{\tilde{\pi}}(x)] = \Lambda_{\tilde{\pi}}(x, \mathcal{T}_x, \gamma)^T \mathrm{Cov}\left[r(\mathcal{T}_x)\right] \Lambda_{\tilde{\pi}}(x, \mathcal{T}_x, \gamma) + \Lambda_{\tilde{\pi}}(x, \mathcal{T}_x \uplus a_v, \gamma)^T \mathrm{Cov}\left[v(\mathcal{T}_x)\right] \Lambda_{\tilde{\pi}}(x, \mathcal{T}_x \uplus a_v, \gamma)$$

which for leaf nodes simplifies to

$$\mathbb{V}[\hat{Q}_{\tilde{\pi}}(x)] = \mathbb{V}[r(x)] + \gamma^2 \, \mathbb{V}[v(x)].$$

The computation of the vector $\Lambda_{\tilde{\pi}}(x, \mathcal{T}_x, \gamma)$ is not trivial since it requires evaluating the tree evaluation policy for all nodes in the subtree. The evaluation policy is usually also a function of the value estimate or the variance which again requires recursive computation from the leaves up to the root. The variance for each node could be expressed recursively as

$$\mathbb{V}[\hat{Q}_{\tilde{\pi}}(x)] = \mathbb{V}[r(x)] + \gamma^2 \tilde{\pi}(x, \mathcal{A}_v)^T \mathrm{Cov}\left[\hat{Q}_{\tilde{\pi}}(x \uplus \mathcal{A}_v)\right] \tilde{\pi}(x, \mathcal{A}_v)$$

where the covariance matrix would have to be computed as a function of the individual value estimates and their variances. If we assume that the value estimates are independent, the covariance matrix is diagonal and the variance computation simplifies to

$$\mathbb{V}[\hat{Q}_{\tilde{\pi}}(x)] = \mathbb{V}[r(x)] + \gamma^2 \tilde{\pi}(x, \mathcal{A}_v)^{2^T} \, \mathbb{V}[\hat{Q}_{\tilde{\pi}}(x \uplus \mathcal{A}_v)]$$

this is computationally simpler since we do not need to manage any covariance structures.

Depending on its usage, the computation of the value estimate and its variance can either be done on a complete tree or continuously during construction by incorporating it into the backup. The key property allowing efficient continuous computation of the values is that they only depend on the nodes in its subtree. This implies that when a node is modified or a new node is added, only the values of its ancestors need to be recomputed. We can therefore efficiently update the value estimates and variances during the backup stage.

Algorithm 5 shows how the backup is modified for general tree evaluators under the independent value estimate assumption. Node.vVar and Node.rVar are estimates of the variance of the value estimate and the reward respectively. For deterministic environments, Node.rVar is 0. Options for estimating Node.vVar will be discussed in the next section but a fixed positive real value tuned as a hyperparameter is a viable option. In the Algorithm description, **dot** is the dot product of two vectors, $\tilde{\pi}$ is a function that returns a vector of the

tree evaluation policy with a_v as the first element. The square of a vector is the element-wise square of the vector and ++ concatenates two vectors.

Algorithm 5 EvaluatorBackup

Require: NewNode, γ, $\tilde{\pi}$
1: Node $\leftarrow$ NewNode
2: **while** Node is not None **do**
3: RealizedEvaluator $\leftarrow$ $\tilde{\pi}$(Node)
4: ChildQVector $\leftarrow$ [Node.v] ++ [Child.Q for Child in Node.Children]
5: ChildVarVector $\leftarrow$ [Node.vVar] ++ [Child.Var for Child in Node.Children]
6: Node.Q $\leftarrow$ Node.r $+ \gamma *$ **dot**(RealizedEvaluator, ChildQVector)
7: Node.Var $\leftarrow$ Node.rVar $+ \gamma^2 *$ **dot**(RealizedEvaluator^2, ChildVarVector)
8: Node $\leftarrow$ Node.Parent
9: **end while**

3.9 AlphaZero

This chapter has previously simply considered general decision trees of random variables. In this section, we will discuss the implications of general tree evaluation on the AlphaZero framework.

3.9.1 Modified Learning

In AlphaZero, the policy and value networks are trained on data from the visitation count policy. If we make changes to the final policy used by the agent it would also make sense to update the learning targets. The target for the policy network would be the policy extracted by the tree evaluator instead of the visitation counts. The target for the value network would also be affected since the collected trajectories are sampled with a different policy.

3.9.2 Epistemic Uncertainty

The new tree evaluators are functions of the value estimates and their estimated variances. The variance in value estimate arises from the randomness of rewards r and simulation values v. We will focus on deterministic environments where the variance of the rewards is zero. For both random rollout MCTS and AlphaZero, the simulation values still have some variance, even for deterministic environments. For random rollout MCTS, the variance arises from the randomness of the rollout policy.

In AlphaZero, the simulation values come from neural network predictions instead. There is no inherent randomness in the network evaluation since if we input the same state twice, it would return the same value. The variance instead arises from epistemic uncertainty [25], which is related to variance in the network parameters. Every time we train the network, the parameters are updated and the network will output different values. These values will also

be affected by exactly what training data the network is trained on which is also sampled in AlphaZero.

Knowledge of the source of uncertainty is important since it might allow us to model the variance more accurately. For example, in most cases, the epistemic uncertainty should be reduced in regions that we have visited and trained extensively on. This could for example be implemented using visitation counts or random network distillation [9].

We could additionally model the covariance between the simulation values. For example, we know that the covariance between the simulation values of two nodes with the same state will be their variance since they are the same random variable. Depending on what state encoding we use, we could also model the covariance as a function of state encoding similarity. This could potentially be implemented using Gaussian process regression [53].

3.9.3 Adding the policy network

So far, we have not discussed the role of the policy network in tree evaluation. Just like the default visitation count evaluator, our proposed evaluators do not explicitly consider the prior policy but the network would still indirectly affect the evaluation if we use policy-guided tree construction like PUCT.

One important question, especially for further research in this direction is how we could better incorporate the policy network in the tree evaluation policies. We start the reasoning process by considering some extreme cases.

- If the simulation budget is zero, so that the only node in the tree is the root node, then the policy network would be the only source of information so the optimal policy must be that of the policy network.
- When the simulation budget is infinite, the information from the policy network is irrelevant since it would pale in significance to the information from the planning tree.

This type of reasoning aligns with a Bayesian perspective where the policy network serves as a prior and as we plan we update our policy belief. One critical question in this context is how much should we trust the prior policy compared to the new observations we receive during planning. In the related work chapter, some previously proposed methods will be presented.

Chapter 4

Related Work

This chapter presents several previously published works related to our proposed method. One of the main motivators for this thesis is the poor performance of visitation counts in the limited simulation budget setting. The first section covers other approaches to combat this issue. The second section focuses on papers quantifying uncertainty in MCTS which is also what we aim to do by estimating variance. The last section includes previous works that modify how Q-values are estimated and propagated up the decision tree. This is something that we also do, particularly when changing the Q-value estimate used in PUCT.

4.1 Limited Simulation Budget

As shown in Section 3.5, the visitation policy approaches the optimal policy when the simulation budget grows large. However, the policy struggles in the limited budget setting [22, 16]. When using random rollouts, the value estimates are usually cheap to compute which allows larger simulation budgets. In the AlphaZero framework, a forward pass through the neural network is required for each simulation which can be time-consuming. This calls for methods better suited for limited budgets. We hypothesize that our proposed new tree evaluators will handle the low simulation budgets better than the visitation count evaluator but there also exists other proposed methods for handling this issue.

In their work, *Monte-Carlo tree search as regularized policy optimization*, Grill et al. [22] identify similar limitations with the visitation count policy under constrained simulation budgets. They take a unified view of the selection and acting policy and propose a novel policy that captures the goal of the search while avoiding some of the pitfalls of the visitation count policy. The novel policy is defined by

$$\arg\max_{\pi} \bar{Q}^T \pi - \lambda_N \mathrm{KL}(\pi_\theta, \pi)$$

with

$$\lambda_N = c\frac{\sqrt{N}}{N + |\mathcal{A}|}$$

which bears a resemblance to our minimal variance-constrained policy. The main difference is that they constrain the policy to be close to the prior policy while we use the minimal variance policy. Their policy still partly considers variance via λ_N since visitation counts and variance are connected but this is the variance of the root node only, not the children. This policy is therefore still dependent on the tree construction. Consider a binary tree where one action is visited $N-1$ times and the other one once. If we let N grow large then their policy will resemble the greedy policy which simply picks the action with the highest Q-value estimate. This is problematic since one of these Q-values is based on only a single simulation which could have huge variance.

Grill et al. [22] additionally propose using their novel policy as the selection policy instead of PUCT. The primary motivation for this is that since Q-values are estimated by averaging subtree returns this estimate is only unbiased if the trajectories are sampled with the same policy as we use at the root. This is related to our method of changing how Q-values are estimated. Instead of changing how trajectories are sampled to get an unbiased estimator, we change how they are analyzed with the same intention.

Another important recent paper dealing with the limited budget setting is *Policy improvement by planning with Gumbel* by Danihelka et al. [16]. They focus specifically on the issues arising when the simulation budget is smaller than the action space size. They propose a unified framework for tree construction and evaluation built on the idea of policy improvement and sampling without replacement. We will not delve into all the details but the main idea is somewhat similar to the one proposed by Grill et al. [22] in that we interpret the policy network as a prior that we incrementally improve by planning. Danihelka et al. [16] also update the selection policy at non-root nodes to a policy similar to the one at the root to avoid bias in the Q-value estimates when averaging returns.

Since the publication of the previous two policy improvement-focused papers, there has been a range of papers extending these ideas for better learning and performance in certain domains. *Muesli: Combining Improvements in Policy Optimization* by Hessel et al. [23] and *EfficientZero V2: Mastering Discrete and Continuous Control with Limited Data* by Wang et al. [50] are two significant examples.

4.2 Uncertainty Analysis

Our work aims to quantify and use uncertainty estimates for the tree search domain by considering estimator variance. In the paper *Variance Reduction in Monte-Carlo Tree Search*, the authors Veness et al. [49] also consider the bias and variance of value estimates of MCTS nodes. The authors suggest that many previous works focus on bias reduction but few on variance reduction. They present three general techniques that can be incorporated into UCT to reduce the variance.

Another paper on this topic is *Bayesian Inference in Monte-Carlo Tree Search* by Tesauro et al. [45]. The authors propose a framework where Bayesian methods are used to propagate value estimate means and variances through the tree. The estimated means and variances are then used in UCT to partly replace the visitation counts. This is similar to our proposed

modification of UCT. A more recent work in the same domain is *Monte-Carlo tree search with uncertainty propagation via optimal transport* by Dam et al. [15]. They model the value estimates as Gaussian distributions and propagate these up the tree using optimal transport. They similarly use this information to improve tree construction.

The last line of work regarding variance analysis for MCTS considers how we can estimate the variance of simulation values. In the paper *The Second Type of Uncertainty in Monte Carlo Tree Search*, Moerland et al. [33] attempts to incorporate the idea that the variance of value estimate for terminal nodes is zero. They propose a method where information about terminal nodes (subtree depth) is propagated up the tree and used to influence UCT. Similarly, Oren et al. [36] propose a method for estimating the epistemic uncertainty throughout the decision tree. They use the methodology developed to encourage deep exploration of the environment.

4.3 Non Arithmetic Mean Backup

We are not the first to modify how Q-values are estimated and propagated up the decision tree. One early such paper is *Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search* by Coulom [13]. The authors argue that the mean backup operator earlier proposed by Chang et al. [11] is inefficient since it will underestimate the value of the best move. Furthermore, they note that purely backing up the maximum value also causes issues since it is very sensitive to noise in individual samples. Eventually, they propose using an algorithm called Crazy Stone for backup. Keller and Helmert [26] experiment with two novel algorithms called *MaxUCT* and *DP-UCT* which replace the mean backup with more greedy backups. They show that this choice can lead to performance benefits. Furthermore, Feldman and Domshlak [20] investigates the worst-case bounds of both regular UCT and DP-UCT.

There have also been attempts to combine the greedy and the average backup. Dam et al. [14] propose a method called *Power-UCT* parameterized by p. Using p they tune which power-mean should be used for estimating Q-values. Setting p to 1 returns the regular arithmetic average UCT while increasing it makes it more greedy and closer to MaxUCT. Similarly, Lanctot et al. [30] propose estimating Q-values by a weighted average of the maximum and mean values. More recently, Willemsen et al. [52] investigated different value targets for AlphaZero in which the greediness of the backup plays a role.

Chapter 5

Experimental Setup

In Chapter 3, we proposed a novel perspective on general tree evaluation. To demonstrate the usefulness of this perspective, we conduct a series of experiments. All experimental code is available at github.com/albinjal/GeneralAlphaZero.

5.1 Agents

This section covers the three different agents compared in the experiments. All agents are compared with the same set of hyperparameters for each environment. The hyperparameters are tuned experimentally so that the baseline agent performs sufficiently well in the Cliff Walking environment. The tuning was assisted by the Weights and Biases platform [6] as well as Bayesian optimization [32]. Exact details on the hyperparameters used are covered in Appendix A.

5.1.1 AlphaZero Agent (Visit+PUCT)

AlphaZero serves as the baseline agent in all experiments. The algorithm is described in Section 2.3. Q-values and final acting policies are determined using the default visitation count evaluator. Tree construction is done using standard PUCT with $\bar{Q}$ unless otherwise specified [38].

Despite other varieties existing, we argue that the AlphaZero/MuZero with Visit+PUCT is the most well-established algorithm which makes it a good baseline. It is also used as the only baseline in Grill et al. [22] and Danihelka et al. [16].

5.1.2 MVC Tree Evaluation Agent (MVC+PUCT)

To demonstrate the use of general tree evaluation, we investigate the performance of the Mean-Variance Constrained tree evaluator presented in Section 3.6. The agent uses the same tree construction algorithm as the AlphaZero agent but we modify **ExtractPolicy** in Algorithm 1 to use the MVC evaluator instead of visitation count.

The MVC evaluator requires an estimate of the variance of simulation values. The experiments are conducted using the assumption that the variance is constant (see Section 3.4). The variance of observed rewards is set to 0 for deterministic environments.

5.1.3 MVC Tree Construction Agent (MVC+MVCPUCT)

In addition to modifying how the final policy is extracted from the built tree, we also propose utilizing the tree evaluation framework during tree construction (Section 3.7). This agent also uses the MVC evaluator for policy extraction but additionally modifies how Q-values in PUCT are estimated. Specifically, equation 3.16 ($\hat{Q}_{\tilde{\pi}_{\text{MVC}}}$) is used instead of equation 3.15 ($\bar{Q}$). Algorithm 5 is used to achieve computational efficiency matching that of the baseline agent. The variance assumptions are the same as for the Tree Evaluation Agent.

5.2 Environments

The codebase developed for this project is implemented with the environment interface of OpenAI Gym in mind [7, 46]. Experiments can therefore be conducted on any of these environments without much modification, except for hyperparameter tuning. The algorithmic modifications we propose primarily focus on how we estimate the values of nodes in the tree. Therefore, we choose to evaluate the agents in environments with simpler reward structures. We also avoid introducing any deep exploration strategies since it is not central to this work. This leads us to avoid environments such as Mountain Car [34] with adversarial dynamics or sparse rewards. We choose to experiment with the Cliff Walking and Frozen Lake environments.

5.2.1 Cliff Walking

Cliff Walking is a grid world environment included in Gym as CliffWalking-v0. It is credited to Sutton and Barto [43]. The goal of the agent is to navigate from the starting state to the goal state as quickly as possible while avoiding jumping into the cliff. The environment is deterministic with a discrete action and state space. The possible actions are moving up, down, left, or right. If the agent moves towards the edge of the environment, it stays in the same state. The agent receives a reward of -1 for each step taken. Stepping into the cliff resets the agent to the starting state and yields a reward of -100. No reward is given for reaching the goal state but this is the only terminal state. The environment is visualized in Figure 5.1.

We constrain the episode length to a finite time horizon $H = 100$. We use a discount factor of $\gamma = 1$ (no discount) since the agent is already incentivized by the reward structure to reach the goal as quickly as possible. The default grid size is 4×12 but we modify it to 6×12. The reason for this will be evident in the next chapter but in essence, it enables additional possible trajectories from start to goal. This is useful for analyzing agent behavior further.

Figure 5.1: 6×12 Cliff Walking Environment. The starting state is in the bottom left corner and the goal state is in the bottom right corner. The cliff is located in the bottom row.

5.2.2 Frozen Lake

Frozen Lake is another grid world environment included in Gym as FrozenLake-v1. The goal for the agent is to navigate from the top right corner to the bottom right without falling into one of the holes. The action space is the same as for the Cliff Walking environment. One major difference between the Frozen Lake and the Cliff Walk is that the holes are terminal states. The reward signals given to the agent are $+1$ for reaching the goal and -1 for falling into a hole. A negative reward for falling into the holes is a modification to the original environment we introduce to prevent an untrained agent from being invariant to jumping into the holes.

To incentivize the agent to walk to the goal as quickly as possible, we apply a discount factor (specified in Appendix A). We use the deterministic version of the environment (no slipping) and constrain the episode length to a finite time horizon $H = 100$. Our experiments are conducted on the default 4×4 and 8×8 map layouts. See Figure 5.2 for a visualization and exact map layout.

5.3 Value Functions

We evaluate the agents using two value function configurations. These are the default AlphaZero setting and the heuristic value function setting. Another potential setting would be the random rollout setting like classical MCTS but we decided to exclude it. The reason for this is that the uniform policy is a poor estimator of the optimal value for the environments in question. Random rollouts are also stochastic which makes the results difficult to interpret.

(a) 4x4 (b) 8x8

Figure 5.2: Frozen Lake Environments. The starting state is in the top left corner and the goal state is in the bottom right corner.

5.3.1 AlphaZero

In the AlphaZero setting, the value and policy functions used in tree search are learned through self-play (see Section 2.3). The learning is dependent on several hyperparameters such as learning rate and neural network architecture. Details on these used for the experiments are covered in Appendix A. Good performance in this setting is important since it is critical for most practical applications. In this setting, the agent iterates between training and evaluation mode.

Training and Evaluation

In the AlphaZero framework, exploration during training is primarily achieved through two mechanisms. Firstly, when collecting trajectories, we sample from the final policy distribution without applying PolicyMax (equation 2.2). This greatly increases the exploration and diversity in training data but might not yield the highest possible expected returns we could achieve. Secondly, AlphaZero adds Dirichlet noise to the prior probabilities of the root node. This also encourages exploration as well as a wider search at the root.

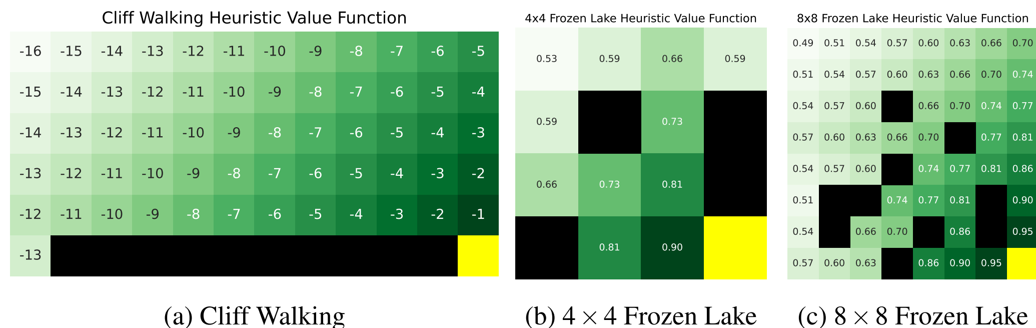
To properly evaluate the best-case performance of the agent, we introduce an evaluation mode where we attempt to avoid exploration. In this mode, PolicyMax is applied to the final policies extracted from the tree. We could also remove the Dirichlet noise completely but we choose to substitute it by mixing in a uniform distribution. This will avoid random exploration without losing the benefit of a wider search at the root. In general, the agents behave almost (ties in PolicyMax is the exception) deterministically during evaluation.

5.3.2 Heuristic Value Function

In the heuristic value function setting, the agent is provided with a value function instead of letting it learn one itself. There are several reasons for doing this. Firstly, our proposed framework focuses on the tree search component of AlphaZero so excluding the learning component further isolates exactly the modified component. Secondly, the heuristic value function is simpler and therefore also easier to analyze. Thirdly, evaluating performance without learning is much faster which allows for more statistically significant results. Lastly, in some practical applications such as Chess engines, heuristically constructed value functions are still used since they can be much faster to evaluate [10, 17].

In this setting, the agent can only access the value function but no prior policy. Therefore UCT is used for tree construction instead of PUCT. Before final action selection, PolicyMax is applied (evaluation mode).

The heuristic value functions used for these environments are the optimal value function for each environment. For all environments, the optimal value function was determined by first determining each state's distance from the goal via breadth-first search. For Cliff Walking, the value is the negative distance and for the frozen lakes, the distance is one times the discount factor to the power of the distance (optimal discounted return). The value functions are visualized in Figure 5.3.



(a) Cliff Walking (b) 4×4 Frozen Lake (c) 8×8 Frozen Lake

Figure 5.3: Heuristic Value Functions used for the Cliff Walking and Frozen Lake.

Chapter 6

Results

This chapter presents the empirical results from the experiments conducted. The results are divided among the Cliff Walking and Frozen Lake environments. The analysis of the Cliff Walking environment is more detailed and discussed since many of the observations are transferable to the Frozen Lake environment. Some additional results are included in Appendix B. The text accompanying the figures is intended to help interpret the results and provide context. The main discussion and conclusions from the results are presented in Chapter 7 and Chapter 8 respectively.

6.1 Cliff Walking

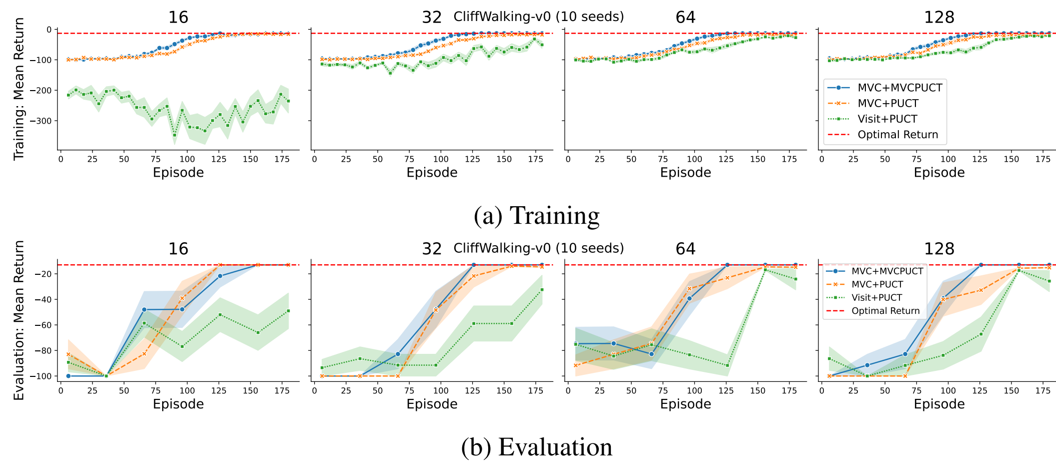


Figure 6.1: Cliff Walking. Training and Evaluation mean return over training episodes. The column numbers indicate the simulation budget.

Figure 6.1 shows the training and evaluation curves for the three agents described in Section 5.1 on the Cliff Walking environment. The horizontal axis describes the number of episodes (trajectories) that the agent has experienced. The vertical axis shows the mean discounted return of the agent. The shaded area indicates the standard error over training

seeds. The dashed red line displays the highest possible discounted return achievable for this environment (-13).

This is a quantitative plot indicating that our two proposed agents outperform the baseline AlphaZero agent. The difference is especially prominent for lower simulation budgets. The difference is less pronounced during evaluation than during training but still significant. To better understand these results, we will discuss the observed behavior with the support of additional data.

6.1.1 Simulation Budget and Entropy

Firstly, we observe that the performance of the baseline AlphaZero agent is significantly lower during training, especially for low budgets. We hypothesize that this is due to the effects described by Grill et al. [22] (section 4.1). Particularly, even if the agent discovers more or less promising actions, it will take additional simulations for this information to be reflected in the visitation counts. In this particular environment, this is determinantal since it also means that if the agent is standing next to the cliff, it has at least a $1 \div \text{budget}$ probability of jumping into the cliff. The reward for falling into the cliff is -100 which is a significant penalty. This will make the baseline agent avoid the cliff which hinders learning and performance since the goal is also located next to the cliff. The MVC agents do not suffer from this issue in the same manner since they take the Q-value estimates into account when selecting actions.

We can observe this effect in the data since for the lower budgets, AlphaZero reports returns less than -100 which is the minimum return when not jumping into the cliff. We note that none of the agents jump into the cliff during the evaluation stage. This makes sense since a PolicyMax is applied before action selection but the performance is still worse. This is likely due to: 1. The policy and value functions learned by the agents are suboptimal and 2. Values are still propagated up the tree with the visitation count evaluator $\bar{Q} = \hat{Q}_{\tilde{\pi}_N}$. This hypothesis is additionally supported by the baseline reporting higher entropies than the MVC agents during training (Figure 6.2). This is interpreted as the baseline acting more randomly during training.

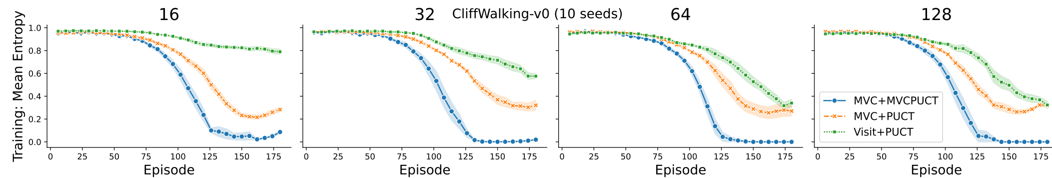


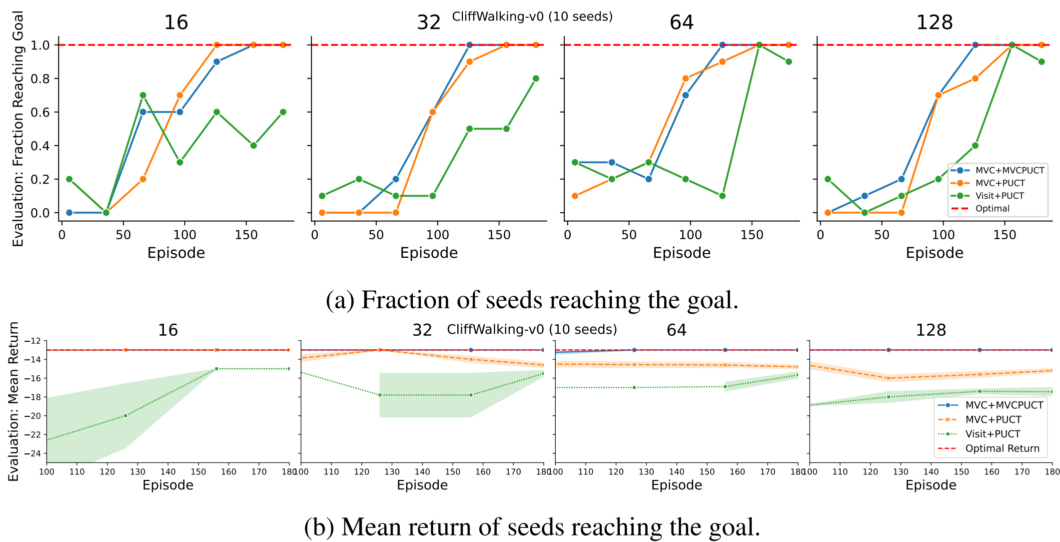
Figure 6.2: Policy normalized entropy over training episodes. 1 indicates uniform distribution and 0 fully deterministic policy.

6.1.2 Completion Ratio and Completion Return

We observe that the evaluation episode outcomes can be partitioned into two different scenarios. Either, the agent finds a path to the goal and terminates the episode or it times out

and returns -100. In the evaluation setting, it is possible that agents get stuck and time out since they act deterministically in each state. If the agent takes actions that lead to reaching the same state again it will get stuck in an infinite loop. For example, if the agent moves against the edge of the environment, it will not change its state and there is no way that the agent will ever reach the goal.

With this knowledge, we can investigate the performance of the agents with two new questions. Firstly, how many of the agents reach the goal before timing out? Secondly, for the agents who find the goal, how quickly do they do it?



(a) Fraction of seeds reaching the goal.

(b) Mean return of seeds reaching the goal.

Figure 6.3: Partition into evaluation completion ratio and completion return. The column numbers indicate the simulation budget.

The first question is answered in Figure 6.3a which displays the fraction of seeds that reach the goal during each evaluation. The MVC agents consistently reach the goal for all simulation budgets. On the other hand, the baseline agent has trouble reaching the goal for lower budgets. For higher budgets, a majority reach the goal but the stability seems worse than the MVC agents.

The second question is answered in Figure 6.3b which shows the mean return of the episodes that reach the goal. The construction agent is barely visible since it achieves optimal returns for all budgets. For earlier episodes, this data is noisy since it is based on only a fraction of the total seeds. Note that for this reason, the episode range is adjusted. Even when excluding the seeds not reaching the goal in the mean calculation, we still observe that our proposed agents achieve higher returns than the baseline in general. We can additionally observe that, in particular for larger budgets, the tree construction agent outperforms the evaluation agent.

6.1.3 State Distributions

To improve our understanding of the observed returns, we can investigate the state distributions of the agents to see what paths they learn. For this, we consider the average visitation counts per state during the final evaluation episodes (10 seeds). The results for all budgets are shown in Figure 6.4. Please refer to figure 5.1 for the environment layout.

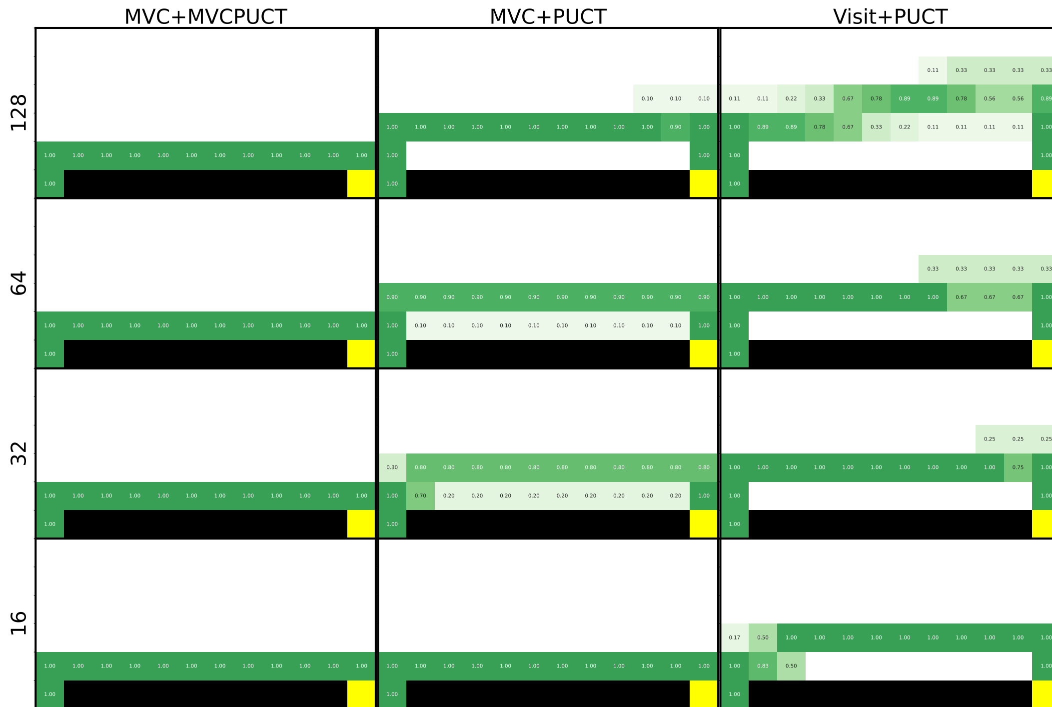


Figure 6.4: Final evaluation state visitation distribution for the Cliff Walking environment. The row numbers indicate the simulation budget and the columns the agents.

The state distributions show that the tree construction agent consistently learns to walk next to the cliff directly to the goal which explains why it has the highest return in Figure 6.3b. The evaluation agent also learns a faster route than the baseline agent but for all budgets except 16, it is not as direct as the tree construction agent. We can also note that, somewhat against our initial intuition, the baseline and tree evaluation agents both seem to avoid the cliff and take longer paths the larger the simulation budget. We will investigate this further by observing what happens in the heuristic value function scenario.

6.1.4 Heuristic Value Function

Figure 6.5 shows the mean returns for the different agents on the Cliffwalking Environment while using the heuristic value function described in Section 5.3.2. The results are collected for 100 seeds for each simulation budget. The primary reason why multiple seeds are collected is because UCT is used as the selection policy. When using PUCT like in the AlphaZero setting, the policy network will dictate the order in which nodes are expanded.

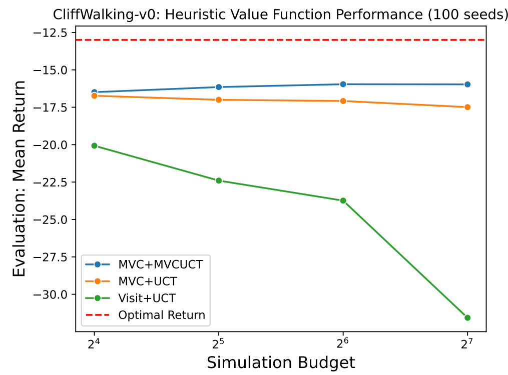


Figure 6.5: Cliff Walking heuristic value function mean return for different simulation budgets (100 seeds).

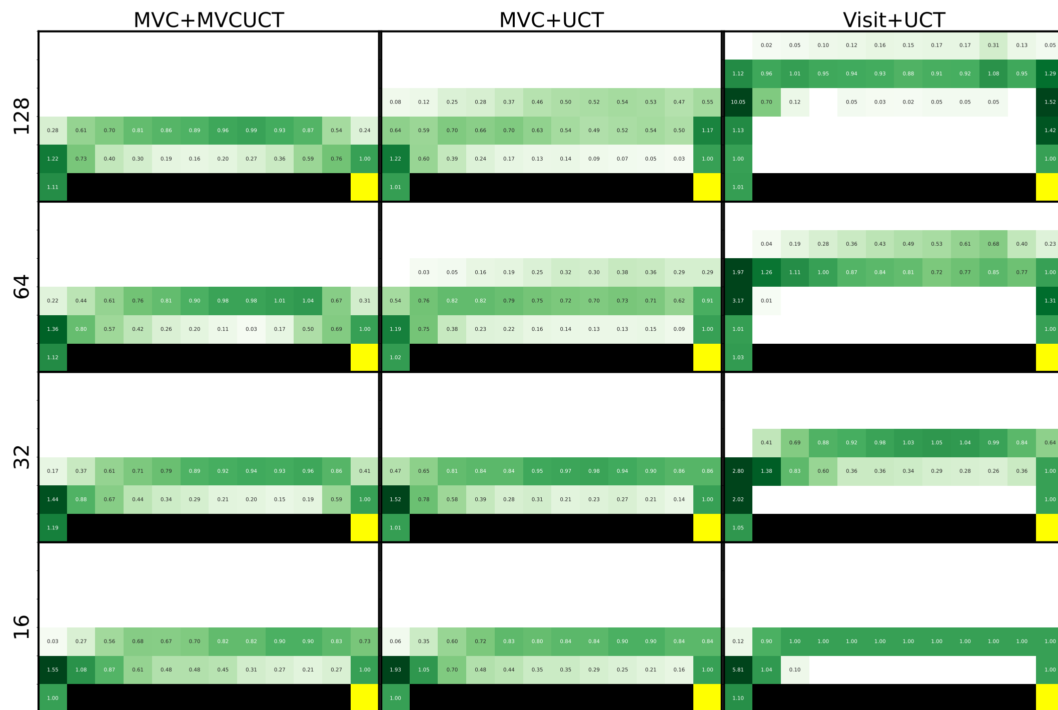


Figure 6.6: Cliff Walking heuristic value function average state density (100 seeds). The row numbers indicate the simulation budget and the columns the agents.

However, in this setting, there is no policy network so the expansion order is randomly sampled from a uniform distribution. In this environment, this has a substantial impact since expanding into the cliff first will discourage further search in this direction.

Figure 6.6 shows the mean state densities over trajectories in the heuristic value function scenario for different budgets. All episodes terminated with the agent reaching the goal (no

timeouts). Similarly to when learning the value function, we observe that the MVC tree construction agent finds the shortest route, followed by the MVC evaluation agent. For the agents constructing the tree with default UCT, we note that the trajectory length increases as the budget increases. The paths also seem generally longer than in the AlphaZero setting. We will reason about this further in the discussion.

6.1.5 Tuning the UCT constant (c)

One potential criticism of the MVC agents is that they come with an additional hyperparameter (β) for tuning. The hyperparameter β balances the weighting of expectation and variance. One could say that it controls the pessimism level. The baseline AlphaZero already comes with the hyperparameter c for tuning the exploration-exploitation trade-off during tree construction (in UCT/PUCT). One could argue that the baseline agent might be able to perform as well as the MVC agents by tuning c. For example, if c is set to a very high value, the visitation count agent will essentially act completely random since the visitations are equally distributed among the children. To investigate this, we conducted experiments in the heuristic value function setting where we tuned the c parameter of the baseline to different extreme values. Note that $c = 2.0$ is what is used in the other Cliff Walking experiments.

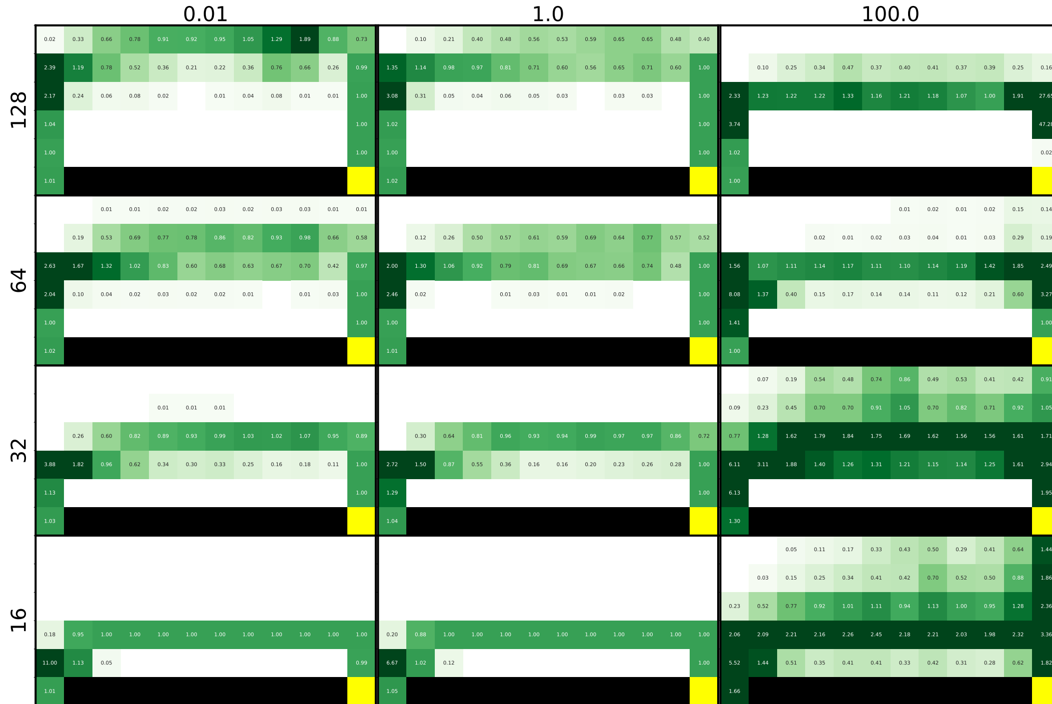


Figure 6.7: Cliff Walking heuristic value function average state density (100 seeds) for baseline (Visit+UCT). The row numbers indicate the simulation budget and the columns the value of the UCT constant c.

The results are shown in Figure 6.7. We observe that the hyperparameter c has an impact on the performance of the baseline agent. However, none of these configurations reach the same performance (short paths) as the MVC agents. As expected, for the large $c = 100$, the agent seems to act mostly randomly for the lower budgets. For higher budgets, the agent's actions seem less random but the agent prefers paths further from the cliff. For the agents with lower values of c, the agent works better for lower budgets but for higher budgets, the lengths of the paths increase. These paths are even longer than the one observed for the agent with $c = 100$. Our hypothesis for this is that with a lower exploration constant, the constructed tree will be deeper which allows the agent to observe the cliff from further away. For a very high c, the tree will be constructed in a close to breadth-first manner which implies that there are not enough simulations to build a deep tree for the actions.

To investigate the effect of c on the structure of the tree, we conduct an additional experiment where we compare the mean state densities of the constructed trees for some selected states. Precisely, for each state, we construct 100 trees of size 128 with $c = 100$ and $c = 1$. We then count the mean number of times each state is observed in the tree and subtract the means to get the difference between the two construction variants. Note that these numbers are not the same as visitation counts for each node since the tree can have many nodes with the same state at different depths.

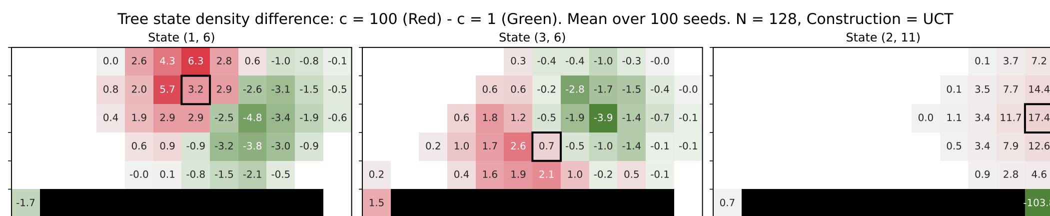


Figure 6.8: Cliff Walking mean tree state counts difference for different UCT constants. Green is for $c = 1$ and red $c = 100$. Each subplot shows a different root state (marked with a black border).

The results of this experiment are shown in Figure 6.8. Note that jumping into the cliff results in instant transportation to the starting state. As expected, the tree constructed with the higher c (Red) seems to have a higher density for states closer to the root state while the low c (Green) constructs a deeper tree toward the goal. This is especially visible in the state (2, 11). We can also observe that for the state (1, 6), the tree constructed with lower c is in general building more towards the cliff and consequently simulating jumping into it more frequently. This can be concluded from the -1.7 observed from the starting state that must come from cliff jumps. These results support our hypothesis that the tree constructed with a lower c will be deeper and observe the cliff from further away.

6.1.6 Tree Construction Differences

We can further investigate the tree state distributions for trees built with regular UCT as opposed to our proposed MVCUCT tree construction. Similarly to Figure 6.8, we construct

trees for 100 seeds with 128 simulations for both the baseline and the MVC tree construction agent. We then calculate the difference in mean state visitation counts for each state.

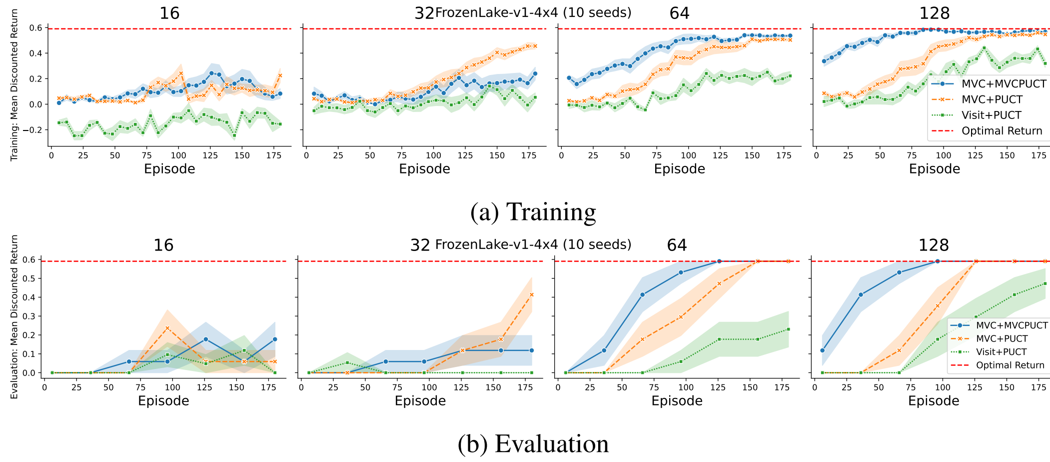
Figure 6.9: Cliff Walking mean tree state counts difference for different construction algorithms. Blue is MVCUCT and green is regular UCT (baseline). Each subplot shows a different root state (marked with a black border).

The results are shown in Figure 6.9. One can see that the modified tree construction has a large impact on the structure of the tree and state distributions. As expected, the modified tree construction is less pessimistic and builds more towards the goal. For state (2, 11) the effect of using the new tree evaluator is similar to the effect of using a lower c which makes some sense since a lower c should also produce a more greedy policy (see Figure 6.8). However, in state (3, 6) one can observe that adjusting c has a different effect than the novel tree construction. MVCUCT builds more towards the goal even if it means going close to the cliff. Conversely, the decreased c builds more away from the cliff. Our hypothesis for this is that the MVCUCT is the only algorithm able to observe the cliff without propagating that negative reward up the tree. Lowering c will make the tree more shallow but it will still propagate the negative reward from observing the cliff which means that search in this direction will be discouraged, even with a more shallow tree. Changing c changes how information is collected while changing the tree evaluator changes how the collected information is analyzed for future decisions.

6.2 Frozen Lakes

Figure 6.10 and Figure 6.11 display the training and evaluation performance in the AlphaZero setting for the 4×4 and 8×8 Frozen Lake environments respectively. The results display similar behaviors as for the Cliff Walking environment. For both map sizes, the visitation counts AlphaZero baseline agent performs poorly in training, especially for lower budgets. For larger budgets, our novel agents reliably optimally solve the 4×4 environment. As expected, the 8×8 environment seems more difficult for all agents. The MVC tree construction agent produces the most promising results in this environment and finds the optimal route for the 64-budget setting. The evaluation performance degrades somewhat for the 128-budget. We hypothesize that this could be mitigated by additional hyperparameter tuning.

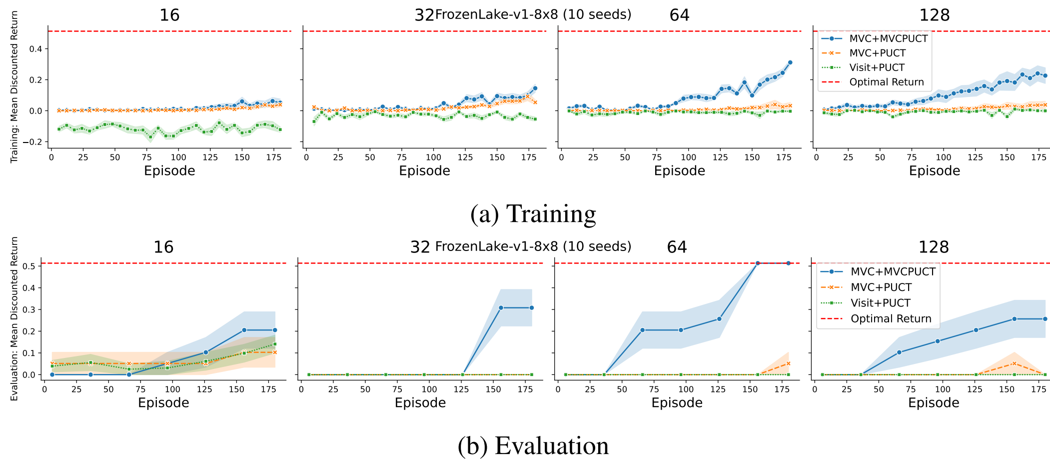
We conduct a similar analysis on the Frozen Lake evaluation state distributions as with the Cliff Walking environment. Figure 6.12 displays the final episode state distributions for all



(a) Training

(b) Evaluation

Figure 6.10: 4×4 Frozen Lake. Training and Evaluation step performance.



(a) Training

(b) Evaluation

Figure 6.11: 8×8 Frozen Lake. Training and Evaluation step performance.

10 seeds. We refer to Appendix B for additional results. We conclude that the most difficult part of the Frozen Lake environment seems to be that the agent has to approach and walk past the holes to reach the goal. This proves especially tricky for the agents constructing the tree with regular PUCT since the initial poor rewards observed during the search will discourage further search in this direction. This leads to the baseline agent often getting stuck in the area around the starting state. This differs from the Cliff Walking environment since the agent can not choose to take an extra-long route to avoid the holes.

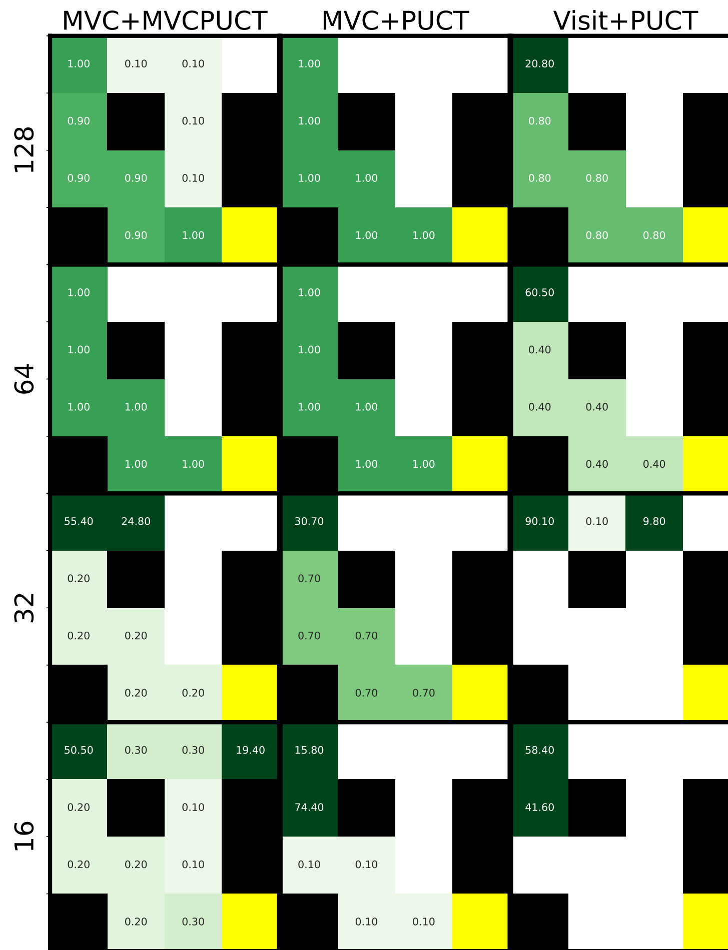


Figure 6.12: Final evaluation state visitation distribution for the 4×4 Frozen lake environment. The row numbers indicate the simulation budget and the columns the agents.

Chapter 7

Discussion

In the analysis section, we proposed a novel framework for general tree evaluation and introduced the MVC tree evaluator. Our experiments show that the two agents based on MVC outperform the baseline agent in both the Cliff Walking and Frozen Lake environments. In this chapter, we further investigate and discuss why this happens.

In our experiments, we operate under the fixed variance assumption which, as we showed in Section 3.5 means that there is a clear connection between visitation counts and variance. In this setting, the visitation count evaluator has a strong resemblance to the minimal variance policy. If we consider the variance bias trade-off for the evaluator, this evaluator would therefore have low variance but underestimation bias. Another way of phrasing this is that it is very pessimistic. On the other hand, the MVC evaluator takes both expectation and variance into account. It can, by tuning beta, express a less pessimistic policy.

We hypothesize that being overly pessimistic is problematic when solving most environments, but especially the Cliff Walk and Frozen Lakes. In the Cliff Walk, the agent will need to leave the left wall and walk closer to the cliff to approach the goal. This proves somewhat adversarial since an overly pessimistic agent might avoid going remotely close to the cliff since it believes it might jump in. Similarly, in the Frozen Lake environment, the agent will have to walk next to holes to reach the goal. If it believes it might jump into a hole it will avoid it and simply never reach the goal. These behaviors can be observed in the reported state distributions.

The issues with overly pessimistic evaluation can be observed both in the AlphaZero learning setting as well as the heuristic value function setting. For the baseline visitation count agent, learning seems to improve with an increased simulation budget. We observe that in the low-budget setting, the visitation count agent might sometimes fall into the cliff/holes during training. We believe this has an immense negative impact on learning since it drastically lowers the value target of the state next to the cliff/hole. Passing that state or nearby states might be necessary to reach the goal.

The difference in evaluation trajectories in the learning or heuristic value function settings highlights another interesting phenomenon. The paths taken to the goal in the learned setting are quicker than those in the heuristic setting, even when using the same hyperparameters.

One explanation for this could be that the value functions learned differ but we believe the main difference arises from the difference between PUCT and UCT. In the learned setting, we use PUCT for searching with a learned policy network. One important task of this network, especially in the low-budget setting is to dictate the order in which leaves are expanded. On the contrary, with UCT, the expansion is done by sampling uniformly. When estimating the value of a tree node adjacent to the cliff, expanding into the cliff will have a profound impact, especially if the visitation count evaluator is used. Observing the large negative reward from the cliff would lower the average return of that subtree and discourage further search in that direction. On the other hand, in the learning setting, the policy network would make sure that the Cliff direction is not expanded first which mitigates this issue.

The experimental results show the importance of tree construction. The baseline visitation count agent as well as the MVC tree evaluation agents are constructed with default UCT/PUCT where the value is estimated by $\bar{Q} = \hat{Q}_{\tilde{\pi}_N}$. We observe that there is a substantial difference in performance for the tree construction agent using $\hat{Q}_{\tilde{\pi}_{\text{MVC}}}$ instead. Even if the tree evaluation agent uses MVC for tree evaluation, it is more pessimistic since the search is guided by $\bar{Q}$. The MVC policy takes both expectation and variance into account to extract a good policy. Since variance is reduced by search, the actions at the root with higher visitation counts will gain additional weight. We believe that this is the reason for the performance difference between the two MVC agents as well as the reason why the difference grows for higher budgets.

When constructing the tree with $\bar{Q}$, exploring the action of falling into the cliff/hole will propagate negative Q-values up the subtree. This leads to this tree generally being expanded away from the cliff/holes which will influence the final policy to do the same. On the other hand, when using $\hat{Q}_{\tilde{\pi}_{\text{MVC}}}$, MVC will assign a lower probability to nodes with low expectation (such as cliff/hole) which means that they will have a low weight in the propagated Q-values. To us, this makes a lot of sense since the probability of the agent jumping into the cliff when adjacent, is near zero, especially in the evaluation setting. This showcases once again how the visitation count evaluator is overly pessimistic. The behavior induced is adversarial for the tested environments which can even lead to worse performance for higher budgets. For low budgets, the decision trees are smaller, and the difference between the two construction algorithms is less impactful.

Another perspective on this issue is that $\hat{Q}_{\tilde{\pi}}$ will propagate Q-values for each node as if it were the root node and we act with $\tilde{\pi}$. It is well established that the visitation count evaluator $\tilde{\pi}_N$ performs poorly in the low-budget setting. Even if we allocate a high budget to the agent, this budget will be distributed among the child nodes which will emulate a lower budget setting from the perspective of these nodes. The dilution of the budget will continue down the tree which means that low-budget settings will be prominent in any MCTS tree. This also helps explain why PUCT seems to outperform UCT, the policy network is especially important for low budgets. The fact that $\hat{Q}_{\tilde{\pi}}$ is applied to every node in the tree highlights the importance of good tree evaluators $\tilde{\pi}$ which can properly assess general trees. It would be interesting to experiment with the policies from Grill et al. [22] and Danihelka et al. [16] since they also claim improved policies for limited simulation budgets.

Chapter 8

Conclusion

In this chapter, we present our conclusions as well as propose directions for future work.

8.1 Conclusion

This thesis is about general tree evaluation for AlphaZero. In Chapter 3 we proposed a novel framework for tree policy extraction and evaluation. The tree node value estimates can be parametrized by tree evaluation policies. The goal of a tree evaluation policy is to reduce the variance of the value estimate without inducing too much bias. Based on this insight, we propose several novel evaluators such as the MVC evaluator balancing variance and expectation. We additionally propose how our insights can be used to improve tree construction.

Our experiments on gym environments demonstrate that our novel framework combined with the MVC evaluator outperforms the AlphaZero baseline, both when learning the value function and when using a heuristic one. Additionally, our suggested modified tree construction algorithm outperforms both agents based on AlphaZero's default construction algorithm. We observe that the effect of the novel tree construction differs from what can be achieved by tuning UCT c-values.

We attribute the success of our novel agents to multiple factors. Our agents do not attempt poor actions such as jumping into a cliff during training. They are not as overly pessimistic as the baseline when propagating values up the tree. Lastly, they do not make any strong assumptions about the structure of the subtrees being analyzed.

Even though our agents outperform AlphaZero, we can not claim that the MVC evaluator achieves state-of-the-art performance in all environments since there are other contenders such as GumbelZero [16] which we did not directly compare to. However, we do believe that our general framework, combined with adjacent improvements in learning, construction, and policy optimization could produce new state-of-the-art agents [50]. Our framework additionally brings major benefits such as being less sensitive to the tree construction algorithm. The final section on future work will cover some specific ideas for further improvements.

8.2 Future Work

This section presents ideas for future work based on our work. We partition the suggestions into three distinct subsections.

8.2.1 Tree Construction

One of the main benefits of a general tree evaluation framework is that it should function regardless how how the tree is constructed. This could pave the way for further improvements in tree construction. One possible direction for this is to reconsider the goal of the selection policy. In literature, the goal is often to minimize the so-called simple or cumulative regret under a multi-armed-bandit assumption [8, 37]. In the context of general tree evaluators, the goal could instead be to select the action with the highest expected new information useful for the evaluator. This would probably yield some optimistic exploration strategy but it would be interesting to analyze this further analytically.

Furthermore, most of the state-of-the-art tree construction algorithms (ex PUCT) operate under the assumption that the tree is expanded one node at a time. This is a problematic restriction since it prohibits parallelization. There have been previous attempts at mitigating this but many of them involve heuristics and produce suboptimal results [44]. This issue is further amplified by the shift towards deep learning-based simulation values. It is well known that batched computation of inputs increases the throughput of neural networks. If we expand nodes one by one (like PUCT), the forward passes are also done with one element at a time. We conjecture that it would be beneficial to expand multiple nodes in each iteration to utilize batched inference. The structure of the tree would be different and overall, the general quality of the tree might be lower. These issues are mitigated by using a general tree evaluator. The benefit is that we could construct much larger trees under the same time constraints which should improve performance.

8.2.2 Variance Analysis

In Chapter 3, we showed that we can use estimator variance instead of visitation counts and utilize this in our proposed tree evaluators. In Section 3.4, we discussed possible assumptions for simplifying variance calculation. In our experiments, we assumed that the variance of the value evaluations was fixed. This is most likely an oversimplification. It would therefore be interesting to experiment further with what can be accomplished under other assumptions. In random rollout MCTS, the rollout depths or sample variance could be considered. For AlphaZero, the epistemic uncertainty could be captured with for example random network distillation or Gaussian process regression (see Section 3.9.2).

Another suggestion related to the variance calculation is to generalize it by estimating full distributions instead. Our purpose in estimating variance and expectation is primarily for constructing probabilistic bounds. We might be able to tighten these bounds with additional information on the distributions. Inspiration for how this can be accomplished can be found in Dam et al. [15]. Another suggestion is estimating and propagating the probabilistic bounds directly.

8.2.3 AlphaZero

Our suggested framework was initially designed with general MCTS trees in mind. The most likely application of this framework is in the AlphaZero setting which utilizes deep learning. We showed experimentally that AlphaZero could benefit from our approach but we conjecture that the performance could be improved further by designing specifically for AlphaZero.

One idea in this direction is to design a tree evaluator that also incorporates the policy network. The output of the policy network should provide some additional information that could be utilized. This was discussed in Section 3.9.3.

Another idea is to utilize this framework to improve the learning targets. In our experiments, we kept the policy and value targets for AlphaZero unchanged. If our proposed framework enables improved estimates of the values of tree nodes, we might be able to use these for the value targets. Additionally, we could consider what would be the most beneficial target for the policy network.

Lastly, we did not consider how our new policies affect environmental exploration. In the conducted experiments, the novel agents did not struggle with this but it could still be beneficial. For example, if one uses random network distillation to estimate the simulation variance and also applies a pessimistic policy for final action selection. In this scenario, the agent would avoid regions it has not yet been in which could be good for evaluation but determinantal for learning. We recommend keeping this under advisement and potentially adding extra incentives for deep exploration such as those proposed by Oren et al. [36].

Bibliography

[1] Ioannis Antonoglou, Julian Schrittwieser, Sherjil Ozair, Thomas K Hubert, and David Silver. Planning in stochastic environments with a learned model. In *International Conference on Learning Representations*, 2021. 1

[2] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multi-armed bandit problem. *Machine learning*, 47:235–256, 2002. 5, 21, 29, 68

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 70

[4] Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957. 3, 4

[5] Sergei Bernstein. On a modification of chebyshev's inequality and of the error formula of laplace. *Ann. Sci. Inst. Sav. Ukraine, Sect. Math*, 1(4):38–49, 1924. 24

[6] Lukas Biewald. Experiment tracking with weights and biases, 2020. URL `https://www.wandb.com/`. Software available from wandb.com. 39

[7] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016. 40

[8] Sébastien Bubeck, Rémi Munos, and Gilles Stoltz. Pure exploration in multi-armed bandits problems. In *Algorithmic Learning Theory: 20th International Conference, ALT 2009, Porto, Portugal, October 3-5, 2009. Proceedings 20*, pages 23–37. Springer, 2009. 58

[9] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018. 33

[10] Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134(1-2):57–83, 2002. 1, 43

[11] Hyeong Soo Chang, Michael C Fu, Jiaqiao Hu, and Steven I Marcus. An adaptive sampling algorithm for solving markov decision processes. *Operations Research*, 53 (1):126–139, 2005. 37

[12] Herman Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, pages 493–507, 1952. 24

[13] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer, 2006. 37

[14] Tuan Dam, Pascal Klink, Carlo D'Eramo, Jan Peters, and Joni Pajarinen. Generalized mean estimation in monte-carlo tree search. *arXiv preprint arXiv:1911.00384*, 2019. 37

[15] Tuan Dam, Pascal Stenger, Lukas Schneider, Joni Pajarinen, Carlo D'Eramo, and Odalric-Ambrym Maillard. Monte-carlo tree search with uncertainty propagation via optimal transport. *arXiv preprint arXiv:2309.10737*, 2023. 37, 58

[16] Ivo Danihelka, Arthur Guez, Julian Schrittwieser, and David Silver. Policy improvement by planning with gumbel. In *International Conference on Learning Representations*, 2021. 1, 18, 30, 35, 36, 39, 56, 57

[17] AD De Groot. Chess playing programs. In *Proceedings Koninklijke Akademie van Wetenschappen, Series A*, volume 67, pages 385–398, 1964. 43

[18] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. Activation functions in deep learning: A comprehensive survey and benchmark. *Neurocomputing*, 503:92–108, 2022. 70

[19] Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J R Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022. 1

[20] Zohar Feldman and Carmel Domshlak. Monte-carlo tree search: To mc or to dp? In *ECAI*, pages 321–326, 2014. 37

[21] Kunihiko Fukushima. Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Transactions on Systems Science and Cybernetics*, 5(4): 322–333, 1969. 70

[22] Jean-Bastien Grill, Florent Altché, Yunhao Tang, Thomas Hubert, Michal Valko, Ioannis Antonoglou, and Rémi Munos. Monte-carlo tree search as regularized policy optimization. In *International Conference on Machine Learning*, pages 3769–3778. PMLR, 2020. 35, 36, 39, 46, 56

[23] Matteo Hessel, Ivo Danihelka, Fabio Viola, Arthur Guez, Simon Schmitt, Laurent Sifre, Theophane Weber, David Silver, and Hado Van Hasselt. Muesli: Combining improvements in policy optimization. In *International conference on machine learning*, pages 4214–4226. PMLR, 2021. 36

[24] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *The collected works of Wassily Hoeffding*, pages 409–426, 1994. 24

[25] Eyke Hüllermeier and Willem Waegeman. Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. *Machine learning*, 110 (3):457–506, 2021. 32

[26] Thomas Keller and Malte Helmert. Trial-based heuristic tree search for finite horizon mdps. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 23, pages 135–143, 2013. 37

[27] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 69

[28] Donald E. Knuth and Ronald W. Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4):293–326, 1975. ISSN 0004-3702. doi: https://doi.org/10.1016/0004-3702(75)90019-3. URL `https://www.sciencedirect.com/science/article/pii/0004370275900193`. 1

[29] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006. 4, 20, 21, 29

[30] Marc Lanctot, Mark HM Winands, Tom Pepels, and Nathan R Sturtevant. Monte carlo tree search with heuristic evaluations using implicit minimax backups. In *2014 IEEE Conference on Computational Intelligence and Games*, pages 1–8. IEEE, 2014. 37

[31] Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017. 1, 8

[32] Jonas Mockus. On bayesian methods for seeking the extremum. In *Proceedings of the IFIP Technical Conference*, pages 400–404, 1974. 39

[33] Thomas M Moerland, Joost Broekens, Aske Plaat, and Catholijn M Jonker. The second type of uncertainty in monte carlo tree search. *arXiv preprint arXiv:2005.09645*, 2020. 37

[34] Andrew William Moore. Efficient memory-based learning for robot control. Technical report, University of Cambridge, Computer Laboratory, 1990. 40

[35] Yazhe Niu, Yuan Pu, Zhenjie Yang, Xueyan Li, Tong Zhou, Jiyuan Ren, Shuai Hu, Hongsheng Li, and Yu Liu. Lightzero: A unified benchmark for monte carlo tree search in general sequential decision scenarios. *Advances in Neural Information Processing Systems*, 36, 2024. 1

[36] Yaniv Oren, Matthijs TJ Spaan, and Wendelin Böhmer. E-mcts: Deep exploration in model-based reinforcement learning by planning with epistemic uncertainty. *arXiv preprint arXiv:2210.13455*, 2022. 37, 59

[37] Tom Pepels, Tristan Cazenave, Mark HM Winands, and Marc Lanctot. Minimizing simple and cumulative regret in monte-carlo tree search. In *Computer Games: Third Workshop on Computer Games, CGW 2014, Held in Conjunction with the 21st European Conference on Artificial Intelligence, ECAI 2014, Prague, Czech Republic, August 18, 2014, Revised Selected Papers 3*, pages 1–15. Springer, 2014. 58

[38] Christopher D Rosin. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61(3):203–230, 2011. 1, 7, 20, 29, 39

[39] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020. 1

[40] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016. ISSN 1476-4687. doi: 10.1038/nature16961. URL `https://doi.org/10.1038/nature16961`. 1, 6

[41] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017. 1, 6

[42] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018. 1, 6, 7, 18, 68

[43] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. 2018. 3, 40

[44] Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki, and Jacek Mańdziuk. Monte carlo tree search: A review of recent modifications and applications. *Artificial Intelligence Review*, 56(3):2497–2562, 2023. 1, 4, 18, 58

[45] Gerald Tesauro, VT Rajan, and Richard Segal. Bayesian inference in monte-carlo tree search. *arXiv preprint arXiv:1203.3519*, 2012. 36

[46] Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, March 2023. URL `https://zenodo.org/record/8127025`. 40

[47] J v. Neumann. Zur theorie der gesellschaftsspiele. *Mathematische annalen*, 100(1): 295–320, 1928. 1, 12

[48] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 1

[49] Joel Veness, Marc Lanctot, and Michael Bowling. Variance reduction in monte-carlo tree search. *Advances in Neural Information Processing Systems*, 24, 2011. 36

[50] Shengjie Wang, Shaohuai Liu, Weirui Ye, Jiacheng You, and Yang Gao. Efficientzero v2: Mastering discrete and continuous control with limited data. *arXiv preprint arXiv:2403.00564*, 2024. 36, 57

[51] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992. 12

[52] Daniel Willemsen, Hendrik Baier, and Michael Kaisers. Value targets in off-policy alphazero: a new greedy backup. *Neural Computing and Applications*, 34(3):1801–1814, 2022. 37

[53] Christopher Williams and Carl Rasmussen. Gaussian processes for regression. *Advances in neural information processing systems*, 8, 1995. 33

Appendix A

Implementation Details

Table A.1: Comprehensive hyperparameters across environments.

Category	Hyperparameter	Cliffwalk	4x4 Lake	8x8 Lake
Planning	UCB c	2.0	1.0	1.0
	UCB value transform	Identity	Identity	Identity
	MVC β	1.0	10.0	10.0
	MVC value transform	Identity	Identity	Identity
	Dirchlect noise α	2.5	2.5	2.5
	Dirchlect noise ε	0.4	0.4	0.4
Environment	Maximum episode length	100	100	100
	Discount factor γ	1	0.9	0.95
Training	Total iterations	30	30	30
	Self-play batch size	6	6	6
	Replay buffer size	90	90	90
	Training batch size	23	23	23
	Epochs per iteration	4	2	4
	Learning rate	0.001	0.001	0.0002
	Learning rate decay	None	None	None
	Optimizer	Adam	Adam	Adam
Loss	State normalized loss	True	True	True
	Value learning n	3	1	2
	Value loss weight	0.7	0.7	0.7
	Policy loss weight	0.3	0.3	0.3
	Regularization loss weight	0.000001	0.000001	0.000001
Neural Network	State embedding	Coordinate	Coordinate	Coordinate
	Architecture type	Separated	Separated	Separated
	Normalization layer	None	None	None
	Hidden dimension	64	64	64
	Hidden layers	2	2	2
	Activation function	ReLU	ReLU	ReLU

The full set of hyperparameters used to produce the results in Chapter 6 are displayed in Table A.1. In the remainder of this section, we will provide further explanations of some of the implementation details and hyperparameters. We divide the section into the categories indicated in the table: Planning, Environment, Training, Loss, and Neural Network.

Planning

The planning hyperparameters are related to how the search trees are constructed and evaluated. The parameter **UCB** c is the exploration constant used in PUCT or UCT (depending on whether we have access to the policy network). The **UCB value transform** and **MVC value transform** are functions applied to the Q-values before usage in UCB or MVC. The point of these is that in literature, it is often assumed that the Q-values are in the range $[0,1]$ [2]. There are many ways in which this transformation can be done, but we choose to leave it out to introduce less complexity. If the normalization is based on empirical Q-values we noticed that the algorithm becomes more sensitive to the initialization of the value network since the noise can be amplified into the zero-one range. Instead, we scale UCB c and the **MVC** β to match the relative Q-value scale of the environments. This is the main reason why the values differ between the Cliff Walk and the Frozen Lake. The return range is wider for the Cliff Walk.

The parameters **Dirchlet noise** α and **Dirchlet noise** ε are used to, during training, add noise to the prior policy in the root node of the search tree. This helps with exploration but is turned off during evaluation. In comparison to what is used in the original AlphaZero paper [42], our values for α and ε are larger. The reason for this is that our environment has smaller action spaces. The final prior policy at the root is, during training calculated as

$$\pi_{\text{prior}}(s,\mathcal{A}) = (1-\varepsilon)\cdot\pi_\theta(s,\mathcal{A}) + \varepsilon\cdot\text{Dirichlet}(\underbrace{[\alpha,\ldots,\alpha]}_{|\mathcal{A}|}).$$

Environment

The **maximum episode length** is the maximum number of steps that can be taken by the agent in each episode. After this, we time out the episode and report the current accumulated discounted reward. The agent can not observe that the end of the episode is near during planning. An episode timing out is treated differently than reaching a terminal state. The **discount factor** γ is the discount factor used in the environment. It is used to weigh future rewards against immediate rewards. We do not use a discount factor in Cliff Walk since there is a negative reward for each step which encourages the agent to finish quickly. In the Frozen Lake environments, we use a discount factor to encourage the agent to reach the goal faster. The agent has access to the discount factor and uses it for improved planning and learning.

Training

The **total iterations** are the total number of self-play + training cycles we train the agent for. The **self-play batch size** is the number of trajectories collected during each self-play stage. These are preferably collected in a batched manner over multiple CPU cores. The total number of episodes is the total iterations times the self-play batch size. We sample uniformly from the replay buffer of size **replay buffer size** to train the agent. The **training batch size** is the number of trajectories we sample from the replay buffer each time. The **epochs per iteration** is the number of times we sample from the buffer and perform a gradient step for each learning iteration. The total number of gradient steps is the total iterations times the epochs per iteration. The **learning rate** is the learning rate used in the optimizer and we do not apply **learning rate decay**, even if this is supported in our implementation. The **optimizer** is the optimizer used for stochastic gradient descent. We use the Adam optimizer [27].

Loss

The **state normalized loss** is a modification to the value and policy loss functions available in our implementation. In the default value and policy losses, the loss is computed as a mean overall environment step in a batch of trajectories. One main difference between our environments and the more classical AlphaZero environments such as chess is that our trajectories can contain many environment steps of the same state. For example, the agent can get stuck in a corner of the grid world until timing out. In this case, a majority of the trajectory states will be the same. To avoid the loss being dominated by these states, we normalize the loss so that the mean is over unique states instead of environment steps. In practice, this is done by first calculating the state counts over the batch and then dividing the step loss by the state counts. The counts are normalized so that the loss magnitude is preserved. Through experiments, we noticed that this improved the performance of all agents.

The **value learning** n is the number of steps we look ahead in the value learning loss. The **value loss weight**, **policy loss weight**, and **regularization loss weight** are the weights used in the final total loss function. These were scaled experimentally so that no component dominate the total loss. The regularization loss is an l_2-norm on all the weights of the neural network.

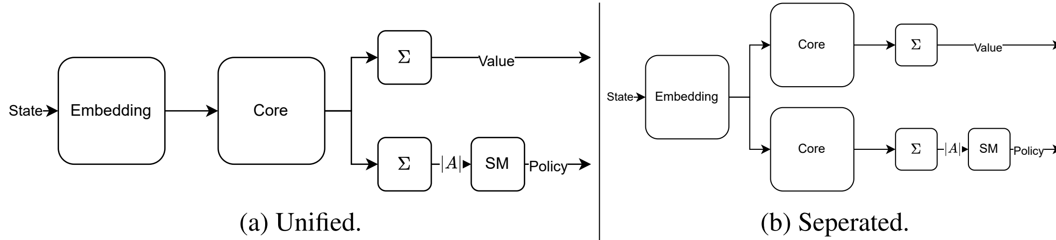
Neural Network

The **state embedding** controls how the environment observations are transformed into feature vectors for the neural network. In the basic discrete gym environments, the state is returned as an integer. The simplest way to embed it is through one hot encoding but our implementation also supports coordinate encoding for the grid worlds which lets the agent generalize further. For the coordinate encoding, the integer is mapped to an (x, y) coordi-

nate. The coordinates are normalized to the range $x, y \in [0,1]$ before being passed to the network.

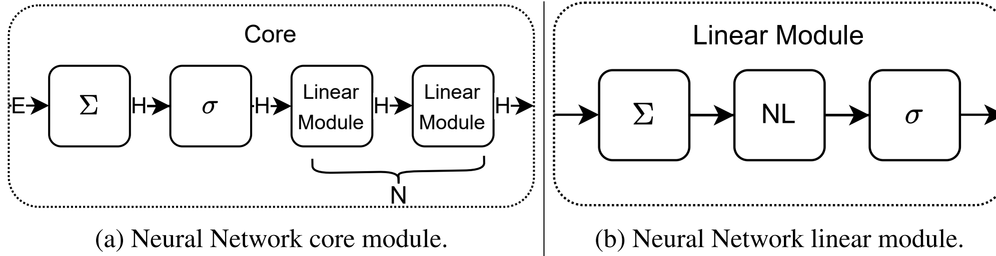
The **architecture type** controls the network structure. The two options are *unified* (Figure A.1a) and *separated* (Figure A.1b). In the figures, Σ indicates a linear layer, *sigma* an activation function, and SM a softmax layer. In the unified structure, the value and policy heads share most of the weights while in the separated structure they do not. When using the unified architecture, the ratio between the value and policy losses should be adjusted properly. We experimented with **normalization layers** such as Layer normalization but found that they did not improve the performance of the agents [3]. The normalization layers are called NL in Figure A.2b.

The **hidden dimension** and **hidden layers** are the size and number of layers in the core module of the neural network. In Figure A.2a, E is the embedding dimension, H is the hidden dimension, and N is the number of Linear Modules (Figure A.2b) in sequence. We used ReLU (rectified linear unit) as the **activation function** in all experiments but also experimented with other functions such as logistic sigmoid [21, 18].



(a) Unified. (b) Seperated.

Figure A.1: Neural Network architecture options.



(a) Neural Network core module. (b) Neural Network linear module.

Figure A.2: Neural Network modules.

Special cases for $\tilde{\pi}(x, a_v)$

In Chapter 3, we presented several tree evaluators derived from optimization problems. The solutions to these problems hold in general but in practice, there are some special cases in implementation. These cases primarily arise when it comes to non-fully expanded nodes and affect the value of $\tilde{\pi}(x, a_v)$. For example, we say that the Q-evaluator is solved by

$$\tilde{\pi}_Q(x,a) = \underset{\mathcal{A}_v}{\text{PolicyMax}} \left[\hat{Q}_{\tilde{\pi}_Q}(x \uplus a) \right].$$

The issue here is that this max is over $\mathcal{A}_v$ which includes a_v. The value of $Q_{\tilde{\pi}_Q}(x \uplus a)$ is by definition $v(x)$. This is a noisy random variable and if this is an overestimation it might be much larger than the value estimate of the child nodes, even if the child value estimate builds on much more data. This would prevent the propagating of information from the subtree to the parent node (only one v is propagated). This is a valid policy but often undesirable since we can not take action a_v in the environment. To circumvent this, most implementations add the condition that $\tilde{\pi}_Q(x, a_v) = 0$ for fully expanded nodes. The same issue can, in a more limited manner, arise for our novel tree evaluators. The issue is more limited since the variance of $v(x)$ will probably decrease its impact. Nonetheless, in our implementation, we decided to simplify this consideration by letting

$$\tilde{\pi}(x, a_v) = \tilde{\pi}_N(x, a_v) = \frac{1}{N(x)}$$

for all tree evaluators. This limits the impact of the described behavior. We believe this to be a reasonable assumption since we use $\tilde{\pi}_N$ as the baseline. This means that we can instead observe the isolated impact of $\tilde{\pi}(x, a)$, $\forall a \in \mathcal{A}$ without having to construct custom rules for $\tilde{\pi}(x, a_v)$ to ensure fair comparisons.

Appendix B

Additional Results

This appendix includes some additional results that were not included in the main text.

Extreme Cliff Walking

In Figure 6.6 we observed that the path lengths of the baseline UCT agent increased when we increased the simulation budget. This sparked the question if this behavior continues forever or if the path lengths would reach some maximum length. To investigate this, we extended the experiment to a simulation budget of $2^{12} = 4096$ for the heuristic value function baseline agent. We also increased the number of rows in the Cliff Walking environment from 6 to 40 to allow the agent to take even longer paths.

The results are displayed in Figure B.1 and Figure B.2. Note that if the agent does not jump into the cliff the return is the negative path length. As expected (observed in Figure 6.6), for lower budgets, the path lengths increase with increased budget. However, for extreme budgets like 1024 and 4096, the pattern breaks and the average lengths decrease again. In Figure B.2, we observe that the distance the agent travels from the cliff stays somewhat constant for higher budgets but it gets more decisive on leaving the left wall. This is likely the primary reason why the total path lengths decrease for higher budgets in Figure B.1.

Frozen Lake Distributions

Figure B.3 and Figure B.5 shows the state visitation distributions in the heuristic value setting for the 4×4 and 8×8 Frozen Lake environments. Figure B.4 shows the final evaluation state visitation distribution for the 8×8 Frozen Lake environment. The distributions are averaged over 100 seeds.

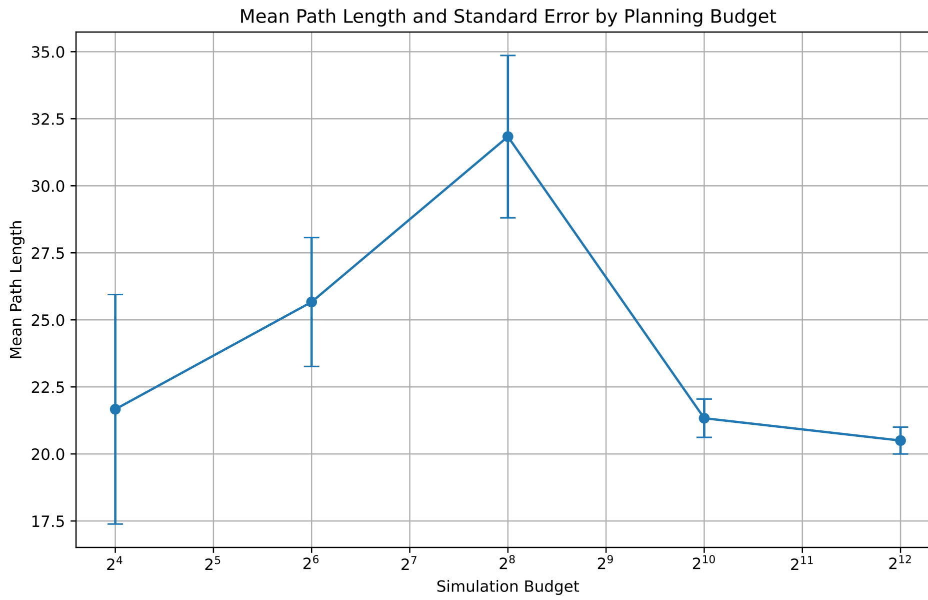


Figure B.1: Cliff Walking 40×12 heuristic value Visit+UCT agent extreme simulation path lengths.

Figure B.2: Cliff Walking 40×12 heuristic value Visit+UCT agent extreme simulation state densities for each budgets. The simulation budgets (from left to right) are 16, 64, 256, 1024, and 4096. Stronger red indicates higher density.

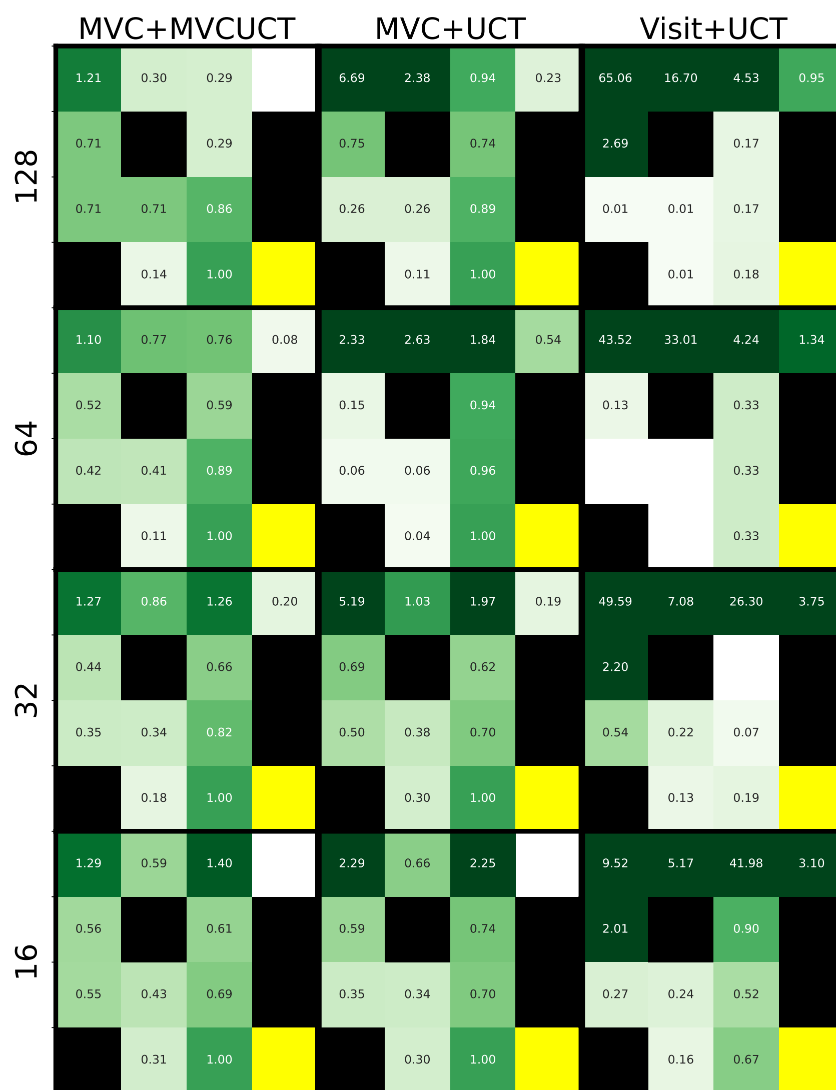


Figure B.3: 4×4 Frozen Lake heuristic value function average state density (100 seeds). The row numbers indicate the simulation budget and the columns the agents.

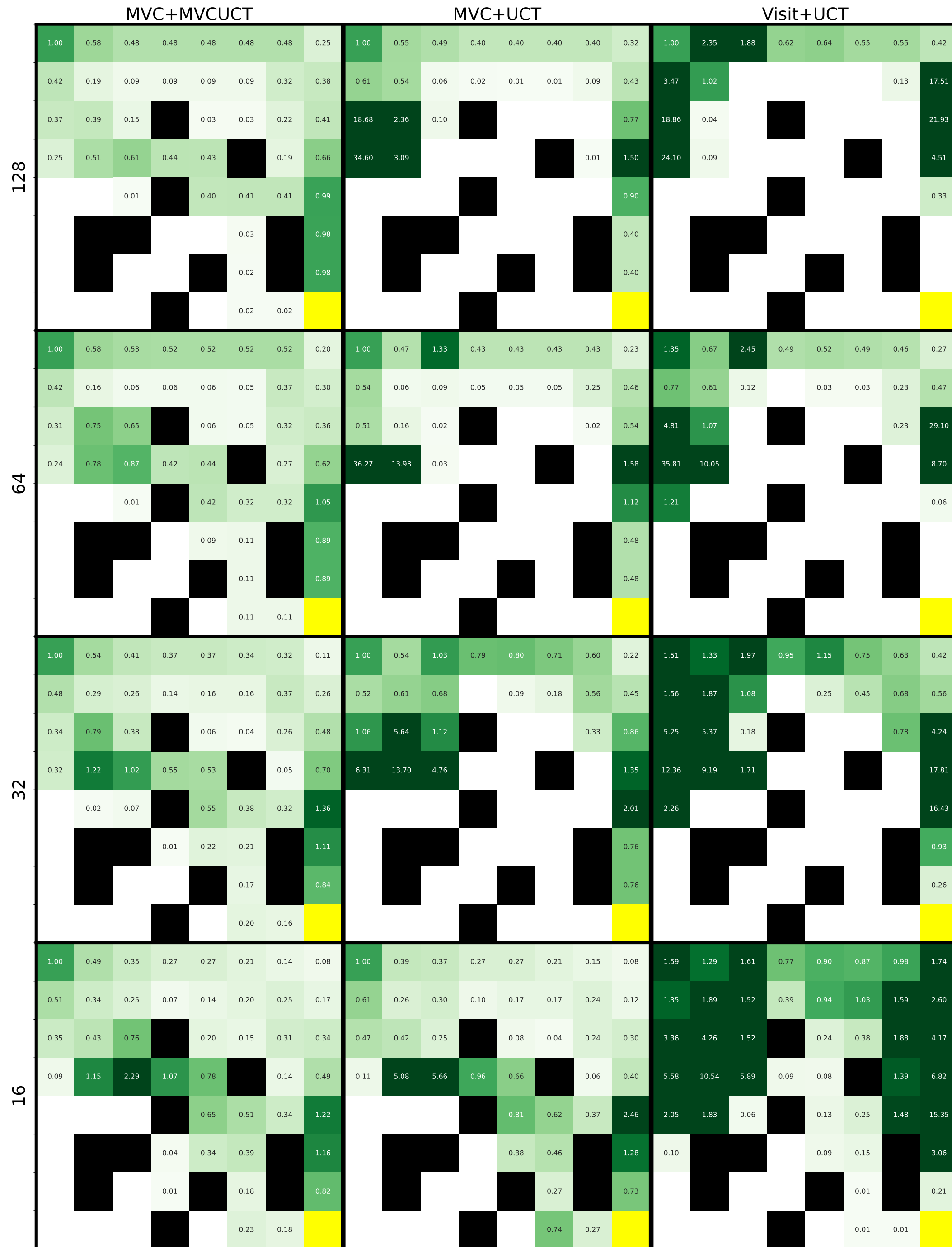


Figure B.4: Final evaluation state visitation distribution for the 8×8 Frozen Lake environment. The row numbers indicate the simulation budget and the columns the agents.

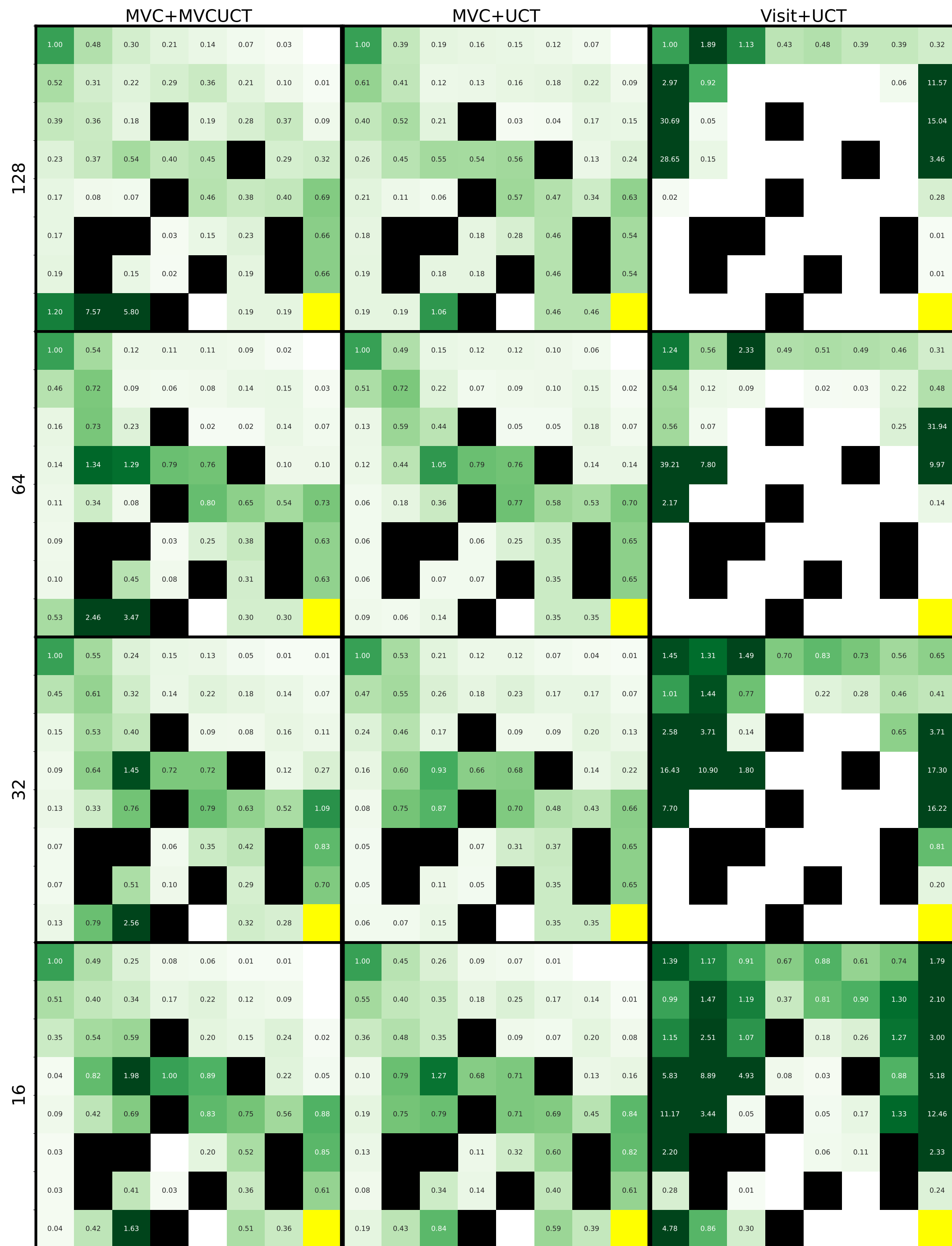


Figure B.5: 8×8 Frozen Lake heuristic value function average state density (100 seeds). The row numbers indicate the simulation budget and the columns the agents.